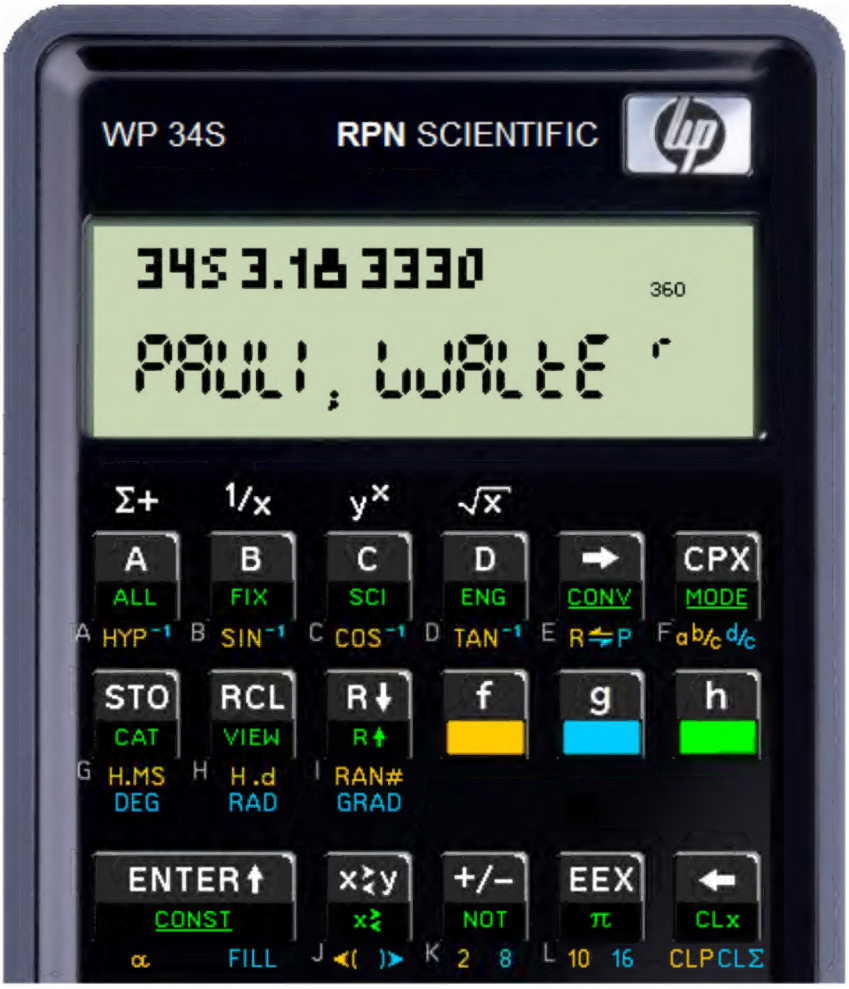
WP 34S OWNER'S MANUAL



This manual documents *WP 34S*. *WP 34S* is free software: you can redistribute it and / or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

WP 34S is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with *WP 34S*. If not, please see *http://www.gnu.org/licenses/* .

This manual contains valuable information. It was designed and written in the hope it will be read thoroughly by you. For really quick and impatient users, however, there is a stand-alone two-page *Troubleshooting Guide* starting on page 180 of this manual.

For those who don't even read this: Sorry, we can't help you.

This manual may change without notice if we, the developers, modify *WP 34S*. We reserve the right to do so at any time. The cover picture may stay unchanged – the document status can be seen from the last note on last page. We recommend you watch *http://sourceforge.net/projects/wp34s/develop* to stay informed.

WP 34S would not have reached its present state without our love for *Classics, Woodstocks, Spices, Nuts, Voyagers*, and *Pioneers*. Thus we want to quote what was printed in *HP* pocket calculator manuals until 1980, so it will not fade:

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sales."

Statement of Corporate Objectives.
Hewlett-Packard

Just in Case …

... you do not have your *WP 34S* calculator yet: *WP 34S* runs on an *HP-20b Business Consultant* or an *HP-30b Business Professional*. Both are financial pocket calculators. So if one of those is sitting on your desk unchanged as produced for *HP*, please turn to *Appendix A* for instructions how to convert it into a full-fledged *WP 34S* yourself. On the other hand, if you do not want to bother with cables on your desk connecting it to your computer, with flashing the calculator firmware and attaching a sticky overlay, you can purchase an *HP-30b*-based *WP 34S* readily on the internet; see e.g. *http://commerce.hpcalc.org/34s.php* or *http://www.thecalculatorstore.com/epages/eb9376.sf/en_GB/?ObjectPath=/Shops/eb9376/Products/%22WP34s%20Pack%22* .

The first way (doing it yourself) will just cost you some time, the second will cost you some money. If you choose buying your *WP 34S* at one of the sites mentioned, we (the developers) will get a very modest fraction of the price to support our otherwise unpaid efforts on the *WP 34S* project. Either way will work – it is your choice.

> Furthermore, there are three optional hardware modifications requiring some fine soldering, cutting, gluing, and drilling a little hole in the plastic case of your business calculator. It is no sorcery for a young electronics engineer with a slow hand and good eyesight, but might come close to it for others (including me, *W.B.*). So check *Appendix H* and decide whether you want to do that yourself. Else look for somebody who is able to do it for you.

The *WP 34S* package includes a full size emulator, so you may test it on your computer before buying any hardware. And later on, you may test your programs there as well before sending them to your *WP 34S*. The function sets are identical.

For the following, we assume the conversion is done and you hold your *WP 34S* in your hands.

TABLE OF CONTENTS

Welcome! 7
Print Conventions and Common Abbreviations 9

Getting Started 10
Keyboard Basics 11
Entering Numbers 15
Real Calculations 16
Elementary Stack Mechanics 17
Error Recovery 21
Calling Commands Unseen on the Keyboard 22

Addressing Objects 24
The Virtual Keyboard for Addressing – Transient Alpha Mode (α_T) 25
Addressing Real Numbers 26
Real Matrices and Vectors 28
Complex Calculations 29
Addressing Complex Numbers 33

Your *WP 34S* in Various Operating Modes 35
The Display 35
Recognizing Calculator Modes 36
Common Commands Returning Specific Displays: STATUS, VERS, ERR, etc. ..38
Floating Point Modes – 1: Introduction and Localisation 39
Floating Point Modes – 2: Displaying Decimal Numbers, Fractions, Times, etc. 40
Floating Point Modes – 3: Real Statistical Calculations 45
Integer Modes – 1: Introduction and Virtual Keyboard 52
Integer Modes – 2: Displaying Integer Numbers 53
Integer Modes – 3: Bitwise Operations 56
Integer Modes – 4: Integer Arithmetic 58
Full Alpha Mode – 1: Introduction and Virtual Keyboard 61
Full Alpha Mode – 2: Displaying Text 64

Programming Your *WP 34S* 66
Labels 67
Tests 67
Local Data 69
Programmed Input and Output, User Interaction and Dialogues 69
Keyboard Codes and Direct Keyboard Access 71
Flash Memory (*FM*) and *XROM* 72

Index of Operations (*IOP*) 74
A 76
B 77
C 78
D 81
E 83
F 85
G 86
H 88
I 88
J 89
K 89
L 89
M 92
N 94
O 95
P 95
R 97
S 100
T 105
U 105
V 106
W 106
X 107
Y 108
Z 108
α 109
β 111
γ 111
δ 111
ε 111
ζ 111
π 111
σ 112
φ 113
χ 113
The Rest 113
Nonprogrammable Control, Clearing and Information Keys 119
Alphanumeric Input 123

Catalogs and Browsers 125
Catalog Contents in Detail 130
Accessing Catalog Items the Fast Way 133
Constants (CONST) 134
Unit Conversions (CONV) 139
Predefined Global Alpha Labels (CAT) 143

The Stopwatch Application 145

Appendix A: Setup and Communication 147
How to Flash Your *HP-20b* or *HP-30b* 147
Overlays – Where to Get Them and How to Make Them Yourself if Required 151
Updating Your *WP 34S* 152
I/O Overview 153

Appendix B: Memory Management 154
Status and Configuration Data 154
Global Registers 154
Summation Registers 156
Subroutine Return Stack (*SRS*) and Program Memory 156
Making Room for Your Needs 157
Addressing and Accessing Local Data, Recursive Programming 158
Switching between Standard Real (*SP*) and Integer Modes 159

Appendix C: Messages and Error Codes 162

Appendix D: The *WP 34S* Emulator on Your Computer 165
Data Transfer between Your *WP 34S* and Your Computer 166
Mapping of Memory Regions to Emulator State Files 168
Simulating a Printer on Your Computer 168

Appendix E: Character Sets 169

Appendix F: Corresponding Operations to the *HP-42S* and *-16C* Function Sets 172
HP-42S 172
HP-16C 178

Appendix G: Troubleshooting Guide 180
Overcoming Troubles in Calculator Operation 180
Overcoming Flashing Troubles 182

Appendix H: Additional Information for Advanced Users 184
Changing Word Size in Integer Modes 184
Mode Storing and Recalling 184
Commands for Advanced Users.....185
Double Precision (*DP*) Calculations and Mode Switching.....186
Further Commands Used in Library and *XROM* Routines 189
Assembler Output 191
Basic Electrical Hardware Specifications of the *HP-20b* and *HP-30b* 191
Tuning the Hardware of Your *WP 34S*.....192

Appendix I: Advanced Mathematical Functions 198
Numbers 198
Statistical Distributions 199
More Statistical Formulas.....204
Orthogonal Polynomials 207
Even More Mathematical Functions 208

Appendix J: Release Notes..... 210

WELCOME!

Dear user, now you have got it: your very own *WP 34S*. It uses the mechanics and hardware of an *HP-20b Business Consultant* or an *HP-30b Business Professional*, respectively, so you benefit from the excellent processor speed of these pocket calculators. And with an *HP-30b* you also get the famous rotate-and-click keys, giving you the tactile feedback that has been appreciated in vintage *Hewlett-Packard* calculators for decades.

On the other hand, the firmware and user interface of your *WP 34S* were thoroughly thought through and discussed by us, newly designed and written from scratch, loaded with functions, pressed into the little memory available, and tested over and over again to give you **a fast and compact scientific calculator like you have never had before** – fully keystroke programmable, comfortably fitting into your shirt pocket, and *RPN*[1].

The function set of your *WP 34S* is based on the famous *HP-42S RPN Scientific* of 1988, the most powerful programmable *RPN* calculator industrially built so far.[2] We expanded the set, incorporating the functions of the renowned computer scientist's *HP-16C*, the fraction mode of the *HP-32SII*, and probability distributions similar to those of the *HP-21S.* We also included **numerous additional useful functions for mathematics, statistics, physics, engineering, programming, I/O, etc.**, such as

- Euler's Beta and Riemann's Zeta functions, Bernoulli and Fibonacci numbers, Lambert's W, the error function, and the Chebyshev, Hermite, Laguerre, and Legendre orthogonal polynomials (no more need to carry heavy printed tables),
- many statistical distributions and their inverses: Poisson, Binomial, Geometric, Cauchy-Lorentz, Exponential, Logistic, Weibull, Lognormal, and Gaussian,
- programmable sums and products, first and second derivatives, solving quadratic equations for real and complex roots,
- testing for primality,
- integer computing in fifteen bases from binary to hexadecimal,
- extended date and time operations and a stopwatch[3] based on a real-time clock,
- financial operations such as mean rate of return and margin calculations,
- 88 conversions, mainly from old *Imperial* to universal *SI* units and vice versa,
- 50 fundamental physical constants as accurate as used today by national standards institutes such as *NIST* or *PTB*, plus a selection of important constants from mathematics, astronomy, and surveying,
- bidirectional serial communication with your computer, as well as printing on an *HP 82240A/B* [4],
- battery-fail-safe on-board backup memory,
- Greek and extended Latin letters covering the languages of almost half of the world's population (upper and lower case in two font sizes), plus mathematical symbols.

---

[1] *RPN* stands for *reverse Polish notation*, a very effective and coherent method of calculating (see p. 15).
[2] The matrix menu of the *HP-42S* cannot be supported by *WP 34S* for hardware reasons. Your *WP 34S* features a set of basic matrix commands and several library routines dealing with matrices instead.
[3] The stopwatch requires adding a quartz crystal and two tiny capacitors.
[4] Printing requires adding an *IR* diode and a resistor. Even a *USB* board is available for your *WP 34S*.

***WP 34S* is the first *RPN* calculator overcoming the limits of a four-level stack** – stop worrying about stack overflow in calculations. *WP 34S* features a choice of two stack sizes expanded by a complex LASTx register: traditional four stack levels for *HP* compatibility, eight levels for convenient calculations in complex domain, for advanced real calculus, vector algebra in 4D, or whatever application you have in mind. You will find a full set of commands for stack handling and navigation in either stack size.

Furthermore, your *WP 34S* features up to 107 global general purpose registers, 112 global user flags, up to 928 program steps in *RAM*, up to 6014 program steps in flash memory, a 30 *byte* alpha register for message generation, 16 local flags as well as up to 144 local registers allowing for recursive programming, and 4 user-programmable hotkeys for your personal favorite functions. Most of the memory layout is conveniently user-settable.

WP 34S is the result of a collaboration of two individuals, an Australian and a German, that started in 2008. We developed *WP 34S* in our free time – so you may call it our hobby (although some people close to us found other names for it). From the very beginning, we have discussed our project in the forum of the *Museum of HP Calculators*. We would like to thank all its international members, who taught us a lot and contributed their ideas and lent their support throughout several stages of our project. Special thanks go to *Marcus von Cube* (Germany) who joined us in bringing *WP 34S* to life by designing an emulator for v1.14, allowing for widespread use and convenient testing. Starting with v1.17, the software began running on an *HP-20b* hardware. A very useful assembler/disassembler has been provided by *Neil Hamilton* (Canada) since v1.18 – even a symbolic preprocessor has been added since v2.1. For v3.0, *Pascal Méheut* (France) contributed a versatile flashing tool for various operating systems. With v3.1, printing on an *HP82240A/B* printer became possible thanks to the gracious support by *Christoph Gießelink* (Germany); a set of micro *USB* boards was developed by *Harald Pott* (Germany); *Ciaran Brady* (UK) wrote a *Beginner's Guide* for our *WP 34S*, and *Christian Tvergaard* and *Peter Murphy* (both USA) carefully proofread this manual. We greatly appreciate all your support!

We named our baby *WP 34S* in honor of the *HP-34C* from 1979, one of the most powerful compact LED pocket calculators. *WP 34S* is our humble approach – within the constraints of *HP's* hardware – to a future *43S* we can only dream of succeeding the *HP-42S* at one point. May our project help to convince those that have access to more resources than we do: Catering to the market for serious scientific instruments is well worthwhile!

We carefully checked all aspects of *WP 34S* to the best of our ability. Our hope is that *WP 34S* is free of severe bugs. However, this cannot be guaranteed. We promise to continue improving *WP 34S* whenever necessary. If you discover any strange results, please report them to us. If they turn out to be caused by internal errors, we will provide you with an update as soon as it is available. Just as before, we will continue striving to maintain short response times.

Enjoy!

Paul Dale and Walter Bonin

Print Conventions and Common Abbreviations

- Throughout this manual, standard font is Arial. Emphasis is added by underlining or **bold** printing. *Specific terms, titles, trademarks, names* or *abbreviations* are printed in italics, *hyperlinks* in blue underlined italics. Bold italic letters such as ***n*** are used for variables; constant **sample** values (e.g. of labels or displayed characters) use bold normal letters. Calculator COMMANDS are generally called by their names, printed in capitals in running text.

- This CPX font is taken for explicit references to calculator keys. Alphanumeric and numeric displays (such as Hello! and 1234) are quoted using the respective calculator fonts where applicable and beneficial.

- Courier is employed for `file names` and numeric `formats`.

- Register **ADDRESSES** are printed using bold Times New Roman capitals, while register ***contents*** are expressed by lower case bold italics. So e.g. the value ***y*** lives in stack register **Y**, ***r45*** in general purpose register **R45**, and ***alpha*** in the alpha register. Overall stack contents are generally quoted in the order [***x***, ***y***, ***z***, …] . Lower case normal italics are for *units*.

All this holds unless stated otherwise.

The following abbreviations are used at various locations in this manual:

DP = double precision (see p. *186*).

FM = flash memory (a special kind of *RAM*, see p. *72*).

IOP = index of operations (see p. *74*).

IR = infrared.

LCD = liquid crystal display.

RAM = random access memory, allowing read and write operations.

ROM = read-only memory, allowing only read operations.

SP = single precision (see p. *159*).

USB = universal serial bus, i.e. a specific interface.

XROM = extended *ROM* (see p. *72*).

Some more abbreviations may be used locally.

Finally: **WARNING** indicates the risk of severe errors. There are only four warnings printed in this manual. Locking up your calculator is the worst that can happen to it, as far as we know.

GETTING STARTED

If you know how to deal with a good old *Hewlett-Packard RPN* scientific calculator, you can start using your *WP 34S* almost right away. Use this manual to get information on some basic design concepts that put your *WP 34S* ahead of previous *RPN* calculators. Continue using this manual for reference.

On the other hand, if this is your first *RPN* scientific calculator at all or the first you use for a long time, we recommend you get an *HP-42S Owner's Manual*. It is available at low cost on a *DVD* distributed by the *Museum of HP Calculators* (see here: *http://www.hpmuseum.org/cd/cddesc.htm*).

Read part 1 of said manual as a starter. It includes an excellent introduction to *RPN*. *RPN* is the reason why your *WP 34S* works without an [=] key; nor does it need the keys [(] nor [)]. Once you got used to *RPN* you may never choose a calculator featuring [=] again.
Part 2 of the *HP-42S Owner's Manual* will help you programming your *WP 34S* for quick and easy solving lengthy, repeated or iterative computations.

Further documentation, including complete information about the other vintage calculators and the famous *PPC ROM* mentioned in this manual, is readily accessible on said *DVD*, too.

Alternatively to the *HP-42S Owner's Manual*, you can download a dedicated *WP 34S Beginners Guide*, recently written by one of our users:
http://sourceforge.net/projects/wp34s/files/doc/WP_34S_Beg_Guide.pdf .

Most traditional commands on your *WP 34S* will work as they did on the *HP-42S*. This little manual here is meant as a supplement presenting all the new features. It contains the necessary information about them, including equations and technical explanations; it is not intended, however, to replace textbooks on mathematics, statistics, physics, engineering, or programming, nor is it a hypothetical *Beginner's Guide to RPN Computing*.

Your *WP 34S* is designed to help you in your calculations and computations. But it is just a tool – although a very powerful one – it cannot think for you nor can it check the sensibility of a problem you apply it to. Do not blame us nor your *WP 34S* for errors you have made. Gather information, think before and while keying in, and check your results: these will remain your responsibilities always.

The following text starts with presenting you the user interface, so you learn where you will find what you are looking for. It continues by demonstrating some basic methods, the *WP 34S* memory and addressing items therein, and the display and indicators that give you feedback about what is going on. Then the major part of this manual consists of an index of all available operations and how to access them, and of lists of all catalog contents. This manual closes with appendices covering special topics, e.g. a list of error messages your *WP 34S* will return if abnormal conditions prevent it from executing your command as expected. There you will also find instructions for keeping your *WP 34S* up-to-date when new firmware revisions appear.

Keyboard Basics

Start exploring your *WP 34S*: Press its bottom left key to turn it on – notice that **ON** is printed below that key. If you turn on your *WP 34S* the very first time, you will get what you see displayed below. To turn it off again, press the green key **h** (notice a little ⊢ showing up top left in display), then **EXIT** (which has **OFF** printed on its lower part). Since your *WP 34S* has *Continuous Memory*, turning it off does not affect the information it contains. To conserve battery energy, your *WP 34S* will shut down some five minutes after you stop using it – when you turn it on again, you can resume your work right where you left off.

To adjust display contrast, hold down **ON** while you press **+** or **−** .

Look at *the keyboard of an HP-42S* for comparison. The most striking difference to it is the colorful surface of your *WP 34S.* You get five functions per key on average. White print on key top is for the *primary* function of the key. For *secondary* functions, green labels are put on the slanted lower faces of 34 keys, golden and blue labels are printed below them on the *key plate.* Grey letters are bottom left of 26 keys.

To access a white label, just press the corresponding key (thus it is called *primary* function). For a golden, blue, or green label, press the *prefix* **f**, **g**, or **h**, respectively, then the corresponding key.

Take the key **5** for **example**. Pressing …

- **5** will enter the digit 5 in display,
- **f** + **5** will calculate the arithmetic mean values of data accumulated in the statistic registers via **x̄**,
- **g** + **5** will compute the standard deviations for the same data via **s**,
- **h** + **5** will open a *catalog* (i.e. a set) of extra statistical functions via **STAT**. All labels printed underlined on the keyboard point to catalogs.
- The grey **R** will become relevant in *alpha mode*, i.e. for input of text.

f, g, and h allow for easily accessing a multiple of the 37 primary functions this hardware can take. The active prefix is indicated by f, g, or h top left in the display for visual feedback, if applicable. You may hold down f, g, or h if you want to call several functions in sequence showing the same color.

Time for a little problem solving **example**. Turn your *WP 34S* on again if necessary (it may have shut down automatically in the meantime). Anyway it will still show its last display

[5].

Now let us assume you want to fence a little rectangular patch of land, 40 *yards* long and 30 *yards* wide.[6] You have already set the first corner post (A), and also the second (B) in a distance of 40 *yards* from A. Where do you set the third and fourth post (C and D) to be sure that the fence will form a proper rectangle? Simply key in

4 0

ENTER↑

This key separates the two numbers in input here.[7]

3 0

→P

Note →P is reached by pressing g + → top right on the keyboard.

So, just get 80 *yards* of rope, nail its one end on post A and its other end on B, fetch the loose loop and walk 30 *yards* away. When both sections of the rope are tightly stretched, stop and place post C there. You may set post D the same way.

This method works for arbitrary rectangles: whatever other distances may apply in your case. As soon as you press →P, your *WP 34S* does the necessary calculation of the diagonal automatically for you. You just provide the land, posts, rope, hammer and nails. And it will be up to you to set the posts!

---

[5] If your *WP 34S* fails to show this display for any reason whatsoever (e.g. because you played around with it a bit in the meantime), you will get it by sliding the battery cover open, locating the little hole below the label RESET, and actuating the button behind it using a suitable pin. Then close the cover again and press ON.

[6] This manual is written for an international readership, and we very well know the *SI* system of units agreed on internationally and adopted by almost all countries on this planet. Despite this fact, we use (old British) *Imperial* units here so our US-American readers can follow. But the example will work with *meters* as well.

[7] Note the number 40 being adjusted to the right and a radix mark added. This indicates input being closed for this number.
Generally, we shall quote only numeric displays in the following for space reasons, using the proper font. And for better readability on paper, we refer to keyboard labels using dark print on white like e.g. EEX or π, omitting the prefix h for the latter since redundant by color print. Also we will use points as radix marks, although significantly less visible than commas, unless specified otherwise explicitly. By experience, "comma people" seem to be more capable of reading radix points and interpreting them correctly than vice versa.

As you have found [→P] on the keyboard next to two other labels showing arrows as well, the labels on your *WP 34S* are generally grouped according to their purposes. Generally, functions and their inverses – if existent – are placed next to each other. If a function is called using [f], its inverse is called using [g]. Besides

- the keys for numeric input ([0], [1], [2], …, [8], [9], [.], [+/-], [EEX], and [←]),
- the three prefixes explained so far ([f], [g], and [h]), and
- the four elementary arithmetic operations ([+], [−], [×], and [/]),

there are five larger groups of labels as outlined below:

Getting used to the groups, you will very soon find your functions on your *WP 34S* easily.

Most of the 168 labels printed on it point to operations carrying simply the same name. [ALL], for instance, calls the function ALL and [FIX] calls FIX. In the example above, however, pressing [→P] called the function →POL – there are a few special labels like this. Let us introduce them to you, starting top left on the keyboard:

1. [A], [B], [C], and [D] are named *hotkeys* – calling directly the user programs carrying these labels. Take your chance to make up to four of your favorite functions directly accessible. Unless the respective labels are defined, these keys act as [Σ+], [1/x], [y^x], or [$\sqrt{x}$], respectively, as is printed above them.

2. [HYP] (i.e. [f] + [A]) is the prefix for hyperbolic functions SINH, COSH, and TANH; [HYP⁻¹] (i.e. [g] + [A]) is the prefix for their inverses ASINH, ACOSH, and ATANH. Similarly, [SIN⁻¹] stands for ASIN, etc.

3. [→] is a prefix for direct conversion of the value displayed (i.e. x). It may be trailed by [H.MS], [H.d], [DEG], [RAD], or [GRAD] (the respective function names then read e.g. →H.MS). [→] trailed by [2], [8], or [16] will display x converted to an integer of the respective base. [→] is also used for indirect register access (see below). [R←] converts polar to rectangular coordinates in a plane, [→P] converts vice versa. So this pair covers the two classic coordinate transformations (see →REC and →POL).

4. [CPX] is for calling complex operations (see p. 29). [a b/c] and [d/c] enter fraction mode for proper or improper fractions (see PROFRC and IMPFRC).

5. [H.MS] and [H.d] represent the classic two time modes, where the latter stands for decimal hours and also for decimal floating point numbers in general (see DECM).

6. [α] sets *alpha mode* (see p. 61)., while [2], [8], [10], or [16] set *integer modes* for calculating with binary, octal, decimal, or hexadecimal numbers (see p. 52).

7. [LG] returns the logarithm for base 10, [LB] does the same for base 2 (see LOG…).

8. [!] calls $x!$ in all numeric modes. [Cy,x] and [Py,x] (calling COMB and PERM here) work like on an *HP-15C*. [r] calls CORR, [|x|] calls ABS, and [RND] calls ROUND.

9. There are three toggles: [./,] for radix marks, [P/R] for programming (like on an *HP-15C*), and [↑] for upper and lower case in alpha mode.

These are all the special labels featured. A complete list of each and every command provided on your *WP 34S*, the keystrokes calling it, and the necessary individual explanation is in the *Index of Operations (IOP)* below for your reference.

Let us return to our introductory example for two remarks[8]:

1. There is no need to enter units in your calculations. Just stay with a consistent set of units and you will get meaningful results within this set.[9] If you want to convert results from one unit to another, however, see the catalog CONV described further below.

2. Although we entered integer numbers only for both sides of our little ground, your *WP 34S* calculated the diagonal in default floating point mode. This allows for decimal fractions of e.g. *yards* in input and output as well. Another mode lets you enter proper fractions such as e.g. 6 ¼ where you need them. Your *WP 34S* features more modes – we will introduce them to you from p. 35 on). Before, let us show you some ways to enter numbers in your *WP 34S* and to further deal with them in typical calculations .

Entering Numbers

... is as easy as typing. For 12.34, for example, press [1][2][.][3][4] . Any digit mistyped may be deleted immediately by [←] and entered correctly thereafter.

For negative numbers such as –5.6 , press [5][+/–][.][6] or [5][.][+/–][6] or [5][.][6][+/–] – the [+/–] changes the sign of the number put in.

For entering really big stuff such as the age of the universe as we know it, do it this way: [1][3][.][8][EEX][9] resulting in 13.8 with nine digits trailing the point, i.e. 13 800 000 000 years. Really tiny numbers such as the diameter of an atom (0.000 000 000 1m – ten zeroes heading the 1) are entered in full analogy: [EEX][+/–][1][0] correspond to $1 \cdot 10^{-10}\,m$.

Any numeric input will just fill the display and is interpreted when completed, not earlier.

[8] Generally, we assume you have graduated from US High School at minimum, passed Abitur, Matura, or an equivalent graduation. So we will not explain basic mathematical rules and concepts here.
And in four decades of scientific pocket calculators, a wealth of funny to sophisticated sample applications has been created and described by different authors – more and better than we can ever invent ourselves. It is not our intention to copy them. Instead, we recommend the DVD mentioned *above* once again: it contains nearly all the user guides, handbooks, and manuals published for vintage *Hewlett-Packard* calculators beginning with their very first, the *HP-9100A* of 1968. Be assured that almost every calculation described there for any scientific calculator can be done significantly faster on your *WP 34S* – and often even in a more elegant way.

[9] The big advantage of *SI* is that it is the largest consistent set available.

Real Calculations

Most of the commands your *WP 34S* features are mathematical operations or functions taking and returning real numbers such as 1 or -2.34 or π or 5.6E-7. Note that integer numbers such as 3, 10, or -1 are just a subset of real numbers.

Many real number functions provided operate on **one number** only. For **example**,

enter [.][4][9] 0.49

and press [√x] . You will get 0.7 since $0.7^2 = 0.49$.

That is easy, isn't it? Generally, such functions replace x (i.e. the value displayed) by its result $\mathbf{f}(x)$. In this example, $\mathbf{f}(x) = \sqrt{x}$.

The vast majority of calculators works this way, so this is no real surprise.

Some of the most popular mathematical functions, however, operate on **two numbers** instead. Think of + and –, for example.

Example: Assume having an account of 1,234 US$ and taking 56.7 US$ away from it. What will remain? One easy way to solve such a task works as follows:

On a piece of paper		On your *WP 34S*		
Write down the 1st number:	1234	Key in the 1st number:	[1][2][3][4]	1234
Start a new line.		Separate 1st from 2nd:	[ENTER↑]	
Write down the 2nd number:	56.7	Key in the 2nd number:	[5][6][.][7]	56.7
Subtract:	1177.3	Subtract:	[−]	1177.3

This is the essence of *RPN*:

Provide the necessary operands, then execute the requested operation.

And a major advantage of *RPN* compared to other entry systems for calculators is that it sticks to this basic rule. Always.[10]

As the paper holds your operands before you calculate manually, a place holding your operands on your *WP 34S* is required. The *stack* does that. It will also take care of intermediate results, if applicable, as your paper may do. Turn overleaf to see how this is done.

[10] Some people claim this being true for *RPL* only. *RPL* is a language developed from *RPN* in the 1980's. Maybe they are even right. In my opinion, however, *RPL* strains the underlying *postfix* principle beyond the pain barrier, exceeding the limit where it becomes annoying for the human brain. Not for everybody, of course, but also for many scientists and engineers. Thus we decided to stick to *RPN* on the *WP 34S*.

Elementary Stack Mechanics

Think of the stack like a pile of registers[11]: bottom up, they are traditionally named **X, Y, Z, T**, optionally followed by **A, B, C**, and **D** on your *WP 34S*. New input is always loaded in **X**, and only its content x is displayed.

Stack register	content
D	d
C	c
B	b
A	a
T	t
Z	z
Y	y
Display **X**	x

[ENTER↑] separates two input numbers by closing the number x and copying it into **Y** [12], so **X** can take another numeric input then without losing information.

After having completed the second numeric input in the little account example above, [−] subtracts x from y and puts the result $\mathbf{f}(x, y) = y - x$ into **X** for display. This method applies for most two-number real functions.

A large part of mathematics is covered by two-number functions of this kind, and combinations of such operations. Let us look at a chain calculation, for **example**

$$\frac{(12.3-45.6)\cdot(78.9+1.2)}{(3.4-5.6)^7}.$$

This is as a combination of six two-number functions: two subtractions, an addition, a product, an exponentiation, and a division. And this is how it is solved on your *WP 34S*, starting top left, and what happens on the stack during the solution: [13]

T				
Z				
Y		12.3	12.3	
X	12.3	12.3	45.6	-33.3
Input	[1][2][.][3]	[ENTER↑]	[4][5][.][6]	[−]

You will have recognized that the first parenthesis was solved exactly as shown in our little account example above. Now proceed to the second parenthesis:

T					
Z		-33.3	-33.3		
Y	A -33.3	78.9	78.9	-33.3	
X	78.9	78.9	1.2	80.1	-2667.33
Input	[7][8][.][9]	[ENTER↑]	[1][.][2]	[+]	[×]

[11] Learn more about the registers provided by your *WP 34S* in next chapter.

[12] This is the classic way ENTER worked from the *HP-35* of 1972 until the *HP-42S* ceased in 1995. It is often said ENTER 'pushes x on the stack'. In doing so, the higher stack contents are lifted out of the way before. So z goes into **T** and y into **Z** before x goes into **Y**. See page 20 for detailed stack pictures. The *HP-30b* employs a different ENTER – the *WP 34S* sticks to classic *RPN*, however.

[13] There may be data loaded in the higher stack levels already. They are from previous operations and are not relevant for this calculation, so leave them aside here.

Note the result of the first parenthesis was lifted automatically (A) to **Y** to avoid overwriting it when the next number was keyed in step 1 of this row. This is called *automatic stack lift* and is standard in *RPN* calculators.[14]

And after having solved the second parenthesis in step 4 of row 2, we had the results of both upper parentheses on the stack – so everything was ready for multiplication to complete the numerator and we did it.

Now we will simply continue and start calculating the denominator:

T						
Z		-2667.33	-2667.33		A -2667.33	
Y	A -2667.33	3.4	3.4	-2667.33	A -2.2	-2667.33
X	3.4	3.4	5.6	-2.2	7	-24943...
Input	3 . 4	ENTER↑	5 . 6	−	7	y^x

Last job remaining is the final division of numerator by denominator. Both are on the stack in the right order. Just press / and see the result: 10.69345346 48 .

As you have observed several times now, the contents of the stack registers drop when a two-number function is executed. Like the automatic stack lift mentioned above, this stack drop affects all levels: x and y are combined giving the result $\mathbf{f}(x, y)$ loaded into **X**, then z drops to **Y**, and t to **Z**. Since there is nothing available above for dropping, the top stack level content will be repeated here. You may employ this top level repetition for some nice tricks. See the following compound interest calculation, for **example**:

Assume the bank pays you 3.25% p.a. on an amount of 15,000 US$; what would be your account status after 2, 3, 5, and 8 years? You are interested in currency values only, so set the display format to FIX 2 for this. It causes the output being shown rounded to next cent (internally, the numbers are kept with far higher precision).[15]

T		1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
Z		1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
Y	10.69...	1.0325	1.0325	1.0325	1.0325	1.0325	1.0325
X	10.3325	1.03	15 UU3	15990.84	16510.55	17601.17	19373.66
Input	1.0325	FILL	15 EEX 3	x x	x	x x	x x x
				after 2 years	... 3 years	... 5 years	... 8 years

Here, each multiplication consumes x and y for the new product put in **X**, followed by z dropping to **Y**, and t to **Z**. Due to top level repetition the interest rate is kept as a constant on the stack, so the accumulated capital value computation becomes a simple series of x strokes.

[14] For a four level stack, the full automatic stack lift procedure is moving z to **T**, then y to **Z**, and finally x to **Y**. The old content of the top level **T** is overwritten. So automatic stack lift affects all stack levels. It will not be indicated after this example anymore. In fact it is worth mentioning when automatic stack lift is disabled since this is under fixed conditions only and occurs far rarer.

[15] In the following, we will use plain text for numeric input for space reasons, unless mentioned otherwise.

Debt calculations are significantly more complicated – so avoid debts whenever possible! In the long run, it is better for you and the economy. Nevertheless, you can cope with those calculations using your *WP 34S* as well (see further below).

There are also a few **three-number real functions** featured by your *WP 34S* (e.g. →DATE and %MRR) replacing x by the result $\mathbf{f}(x, y, z)$. Then t drops into $\mathbf{Y}$ and so on, and the content of the top stack level is repeated twice.

Some real functions (e.g. DECOMP or DATE→) operate on one number but return two or three. Other operations (such as RCL or SUM) do not consume any stack input at all but just return one or two numbers. Then these extra numbers will be pushed on the stack, taking one level per real number.

In addition to the traditional stack and register control operations [ENTER↑], [x⇄y], [R↓], [STO], [RCL], LASTx, CLSTK, [VIEW], [R↑], and [x⇄] – known for decades and mostly found within the blue frame on the keyboard – your *WP 34S* also features [FILL], DROP, RCLS, STOS, SSIZE4, SSIZE8, SSIZE?, Y⇆, Z⇆, T⇆, and ⇆. Turn overleaf to learn what [ENTER↑], [FILL], DROP, [x⇄y], [R↓], [R↑], and LASTx do with the stack if SSIZE4 or SSIZE8 is set. See the *IOP* for information about the other commands.

For the first time ever in a calculator, your *WP 34S* offers you a choice of four or eight stack levels (see SSIZE4 and SSIZE8). Thus, the fate of stack contents depends not only on the particular operation executed and its domain but also on the stack size chosen. Real functions in a four-level stack work as known for decades. In the larger stack of your *WP 34S* everything works alike – just with more levels for intermediate results. Turn overleaf for details.

	Level	Assumed stack contents at the beginning:	Stack contents after executing … … the stack register operations: ENTER	FILL	DROP	x⇆y	R↓	R↑	LASTx	… functions of … one number such as x^2	… two numbers such as /
With **4** stack levels	T	$t = 44.4$	33.3	11.1	44.4	44.4	11.1	33.3	33.3	44.4	44.4
	Z	$z = 33.3$	22.2	11.1	44.4	33.3	44.4	22.2	22.2	33.3	44.4
	Y	$y = 22.2$	11.1	11.1	33.3	11.1	33.3	11.1	11.1	22.2	33.3
	X	$x = 11.1$	11.1	11.1	22.2	22.2	22.2	44.4	last x	123.21	2
With **8** stack levels	D	$d = 88.8$	77.7	11.1	88.8	88.8	11.1	77.7	77.7	88.8	88.8
	C	$c = 77.7$	66.6	11.1	88.8	77.7	88.8	66.6	66.6	77.7	88.8
	B	$b = 66.6$	55.5	11.1	77.7	66.6	77.7	55.5	55.5	66.6	77.7
	A	$a = 55.5$	44.4	11.1	66.6	55.5	66.6	44.4	44.4	55.5	66.6
	T	$t = 44.4$	33.3	11.1	55.5	44.4	55.5	33.3	33.3	44.4	55.5
	Z	$z = 33.3$	22.2	11.1	44.4	33.3	44.4	22.2	22.2	33.3	44.4
	Y	$y = 22.2$	11.1	11.1	33.3	11.1	33.3	11.1	11.1	22.2	33.3
	X	$x = 11.1$	11.1	11.1	22.2	22.2	22.2	88.8	last x	123.21	2

Using the stack, *RPN* makes all parentheses such as [{], [[], [(], [)], []], or [}] completely unnecessary in calculations. There is no operator precedence. Here is another **example** showing a slightly more complicated formula and the keystrokes used for solving it:

$$\sqrt{\frac{1+\left|\left(\frac{30}{7}-7.6\times 0.8\right)^4-\left(\sqrt{5.1}-\frac{6}{5}\right)^2\right|^{0.3}}{\left\{\sin\left[\pi\left(\frac{7}{4}-\frac{5}{6}\right)\right]+1.7\times(6.5+5.9)^{3/7}\right\}^2-3.5}}$$

7 [ENTER] 4 [/] 5 [ENTER] 6 [/] [−] [π] [×] [SIN]	$\sin\left[\pi\left({}^7/_4-{}^5/_6\right)\right]$
6.5 [ENTER] 5.9 [+] 3 [ENTER] 7 [/] [y^x] 1.7 [×] [+]	$\{\sin[\ldots)^{3/7}\}$
[x^2] 3.5 [−]	complete denominator
7.6 [+/−] [ENTER] .8 [×] 30 [ENTER] 7 [/] [+] 4 [y^x]	$(30/7-7.6\times 0.8)^4$
6 [ENTER] 5 [/] 5.1 [√x] [−] [x^2]	$\left(\sqrt{5.1}-6/5\right)^2$
[−] [|x|] .3 [y^x] 1 [+]	complete numerator
[x⇄y] [/] [√x]	complete result (0.37)

Even solving this formula requires only four stack levels as indicated by the colors above. Note there are no *pending operations* – each operation is executed individually, one at a time, allowing perfect control of each and every intermediate result. That is another characteristic advantage of *RPN*.

Calculating such a formula from inside out stays a wise strategy. If you had started with the numerator of that sample formula straight ahead, you would have needed five levels for its complete solution. With eight levels as provided here, you will be on the safe side even with the most advanced equations you compute in your life as a scientist or engineer. **Let your *WP 34S* do the arithmetic while you do the mathematics!**

Error Recovery

Nobody is perfect – errors will happen. With *RPN*, however, error recovery is easy even in long calculations since your *WP 34S* loads x into the special register **L** (for Last x) automatically just before a function is executed.[16] How does this help in real life?

1. If you got an **error message** in response to your function call, press ⬅ to erase that message and you will return to the state before that error happened. Now do it right!

2. If you have erroneously called a wrong **one-number function** (or [Δ%]), just press ⬅ and [RCL][L] [17] to undo it. These two steps restore the stack exactly as it was before the error happened – then resume calculating where you were interrupted.

3. Recovering from wrong **two-number functions** calls requires three steps.

   Example: Assume – while you were watching an attractive fellow student or collaborator – you pressed [×] inadvertently instead of [/] in the second last step of the example on previous page. Murphy's law! Do you have to start the calculation all over now? No, that error is easily undone as follows:

T	older stuff	older stuff	older stuff	older stuff	older stuff	older stuff	older stuff
Z	old stuff	older stuff	old stuff	older stuff	old stuff	older stuff	older stuff
Y	numerator	old stuff	num * den	old stuff	num	old stuff	old stuff
X	denominator	num * den	den	num	den	num/den	0.37
Input		[×]	[RCL][L]	[/]	[RCL][L]	[/]	[√x]
	Fine so far.	**Oops!**	1	2	3	Resume	

   In step 1, [RCL][L] recalls the complete denominator, the last x before the error.

   Step 2 undoes the erroneous operation by executing its inverse.

   Finally, step 3 regains the stack exactly as it was before the mistake. Now you can simply resume your calculation and you will get the correct complete result.

[16] There are only a few commands changing x but not loading **L**. They are mentioned explicitly in the *IOP*.
[17] Note there is a grey **L** printed bottom left of the key [EEX]. Nevertheless, we will quote these keystrokes as [RCL][L] instead of [RCL][EEX] for reasons becoming clear in the next section.

An erroneous [y^x] may be undone the same way:

T	older stuff	older stuff	older stuff	older stuff	older stuff
Z	old stuff	older stuff	old stuff	older stuff	old stuff
Y	num	old stuff	num ^{den}	old stuff	num
X	den	num ^{den}	den	num	den
Input		[y^x]	[RCL][L]	[$^x\sqrt{y}$]	[RCL][L]
		Oops!	1	2	3

… but where do we find [$^x\sqrt{y}$] ? This problem will be solved overleaf.

More advanced two-number functions may require more effort for error recovery. Having called [LOG_x] inadvertently can be undone, too, but needs an additional auxiliary register:

T	older stuff	older stuff	older stuff	older stuff	older stuff	older stuff	older stuff
Z	old stuff	older stuff	old stuff	old stuff	old stuff	older stuff	old stuff
Y	num	old stuff	LOG_{dn}nm	LOG_{dn}nm	den	old stuff	num
X	den	LOG_{dn}nm	den	den	LOG_{dn}nm	num	den
Input		[LOG_x]	[RCL][L]	[STO] 00	[x⇄y]	[y^x]	[RCL] 00
		Oops!	1	2	3	4	5

Generally, this kind of recovery procedure requires that the inverse of the erroneous operation is provided.

4. Inadvertent **pushes on the stack** can be undone by executing DROP once (for [ENTER↑], [RCL], [ŷ], [r], and alike) or twice (e.g. for [x̄] or [s]). You will lose the contents of the top (two) stack level(s) by this mistake, however, so set your stack to eight levels to minimize the effects.

Calling Commands Unseen on the Keyboard

Your *WP 34S* features more than 600 different commands. 168 labels are printed on the keyboard. So how do you learn about the other commands? And when you know their names, how can you call them?

The answer to the first question is most easy: read! The *IOP* contains everything.

The answer to the second question is less obvious, but easy as well: the 'hidden commands' are stored in *catalogs*. Remember labels underlined point to such collections of commands. You will find such labels on your *WP 34S* on the slanted fronts of ten keys. For example, [h] + [3] points to [X.FCN], the largest catalog provided.

Example: Assume you want to learn about DECOMP – it was mentioned on p. *19*. You return to the *Table of Contents* above, look for the *Index of Operations*, jump directly to the letter **D** therein, and look up DECOMP to get the necessary information where it lives and what it does. OK, enter 0.375 and check it – calling such a command is even easier than looking it up:

You have read DECOMP is stored in [X.FCN].

So just key in … and your WP 34S displays …

[X.FCN]	÷ ³√x	being the first command stored in [X.FCN].
[D]	÷ DATE	for obvious reasons.
[▼]	÷ DATE→	
[▼]	÷ DAY	
[▼]	÷ DAYS+	
[▼]	÷ DECOMP	voilà! Now execute it …
[XEQ]	y/x = 8.	
[x≷y]	3.	since 0.375 = 3 / 8 . Expected that. But what is this:
0.46875	0.46875	?
[X.FCN]	÷ DECOMP	oh, good, that function is memorized !
[XEQ]	y/x = 32.	
[x≷y]	15.	meaning 0.46875 = 15 / 32 .

As [▼] browses forwards in a catalog, [▲] does backwards. There is a more elegant method for calling cataloged commands described further below, but this here suffices and will do for the time being.

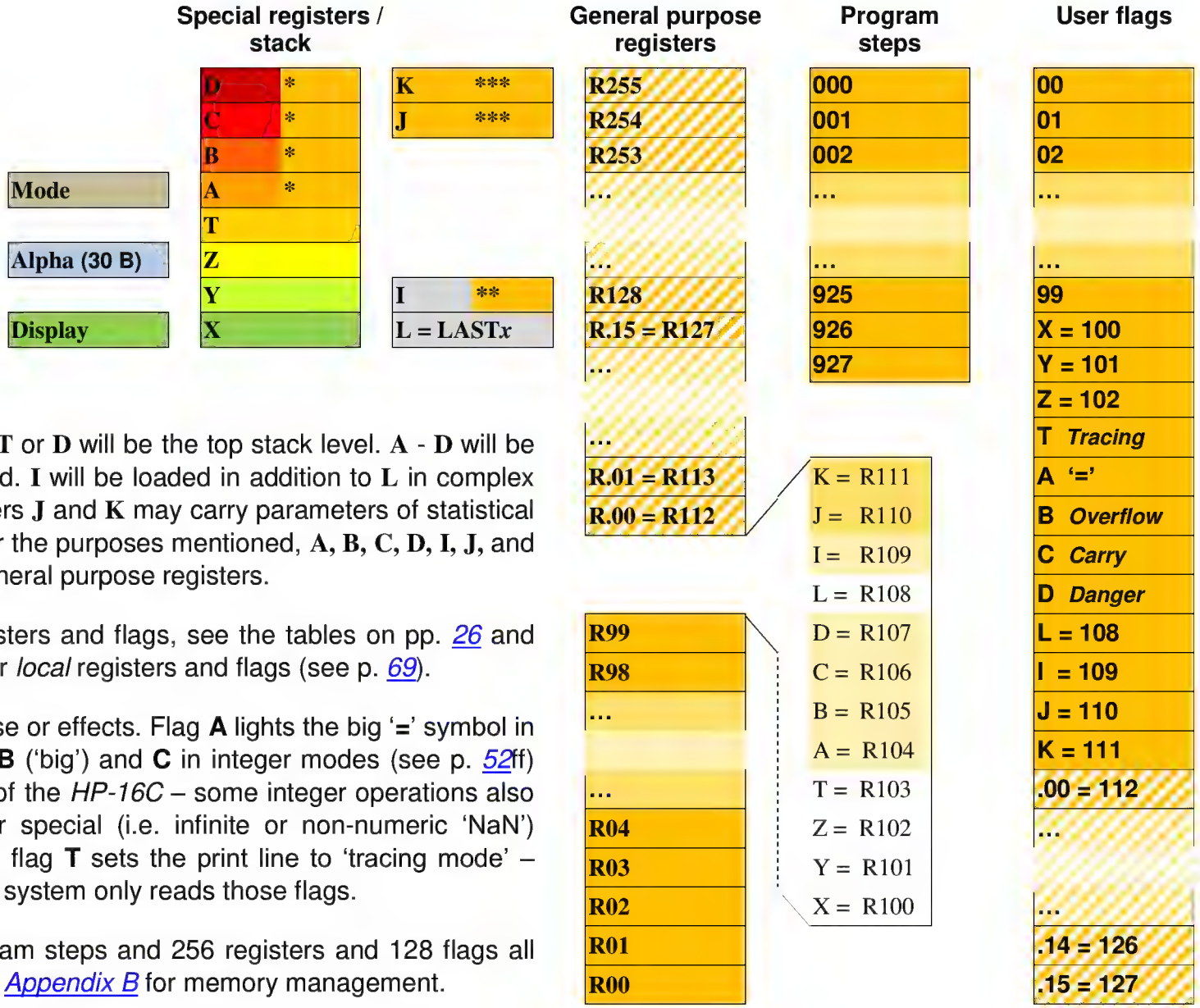
And answering the question of last section, [ˣ√y] is stored in [X.FCN] as well. Just key in [X.FCN][X] [▼][▼][▼] and you get ÷ ˣ√y.

Now you know sufficient about calling commands and real calculations on the stack.

There are, however, far more places in your *WP 34S* where numbers may be stored and saved, and there are also more objects than just real numbers your *WP 34S* can handle for you. Let us show them to you.

ADDRESSING OBJECTS

This picture shows the complete **address space** of your *WP 34S*. Depending on the way you configure its memory, a subset of all these addresses will be accessible.



Offering two stack sizes, either **T** or **D** will be the top stack level. **A** - **D** will be allocated for the stack if required. **I** will be loaded in addition to **L** in complex calculations (see p. 29). Registers **J** and **K** may carry parameters of statistical distributions. Unless required for the purposes mentioned, **A, B, C, D, I, J,** and **K** are available as additional general purpose registers.

For numeric addressing of registers and flags, see the tables on pp. 26 and 33. Addresses ≥112 are used for *local* registers and flags (see p. 69).

Some user flags have special use or effects. Flag **A** lights the big '**=**' symbol in display. The system sets flags **B** ('big') and **C** in integer modes (see p. 52ff) like the overflow and carry bits of the *HP-16C* – some integer operations also read flag **C**. Flag **D** allows for special (i.e. infinite or non-numeric 'NaN') results without getting an error, flag **T** sets the print line to 'tracing mode' – both flags are user-settable, the system only reads those flags.

Note you will not get 928 program steps and 256 registers and 128 flags all together at the same time – see *Appendix B* for memory management.

The Virtual Keyboard for Addressing – Transient Alpha Mode (α_T)

During input processing in memory addressing, e.g. while entering parameters for storing, recalling, swapping, copying, clearing, or comparing, you will need far less than the 168 labels presented on the keyboard. Just 34 keyboard labels will do instead. The calculator mode supporting exactly these 34 labels (and no more) is called *transient alpha mode*. As shown in examples on the next two pages, it may be (temporarily) set in cases as outlined above. Entering α_T mode, the keyboard is automatically reassigned to work as pictured here:

This kind of picture is called a *virtual keyboard* since it deviates from the physical (or real) one of your *WP 34S*. In such a picture, dark red background is used to highlight changed key functionality. White print denotes primary functions also on virtual keyboards, such as the top left key entering the letter A in α_T mode directly. On the other hand, what is printed white on your physical *WP 34S* is called a default primary function.

Note all keys are primary in α_T mode – no shift keys needed. This allows for fast and easy input of a limited character set. So you can reach all register addresses available with a minimum of keystrokes.

Special rules may apply for **T**, **X**, and **Z** here – see two pages below.

α_T mode will be terminated or left (returning to the mode set before) as soon as sufficient characters are put in for the respective step. You may delete pending input (character by character) using ⬅ or just abort the pending command by EXIT – the latter will leave α_T mode immediately.

Addressing Real Numbers

1	User input	x= ?, x≠ ?, x< ?, x≤ ?, x≈ ?, x≥ ?, or x> ?			
	Echo	**OP _ ?** (with α_T mode set), e.g. x>_?			
2	User input	0 or 1	*Stack level or lettered register* Y, Z, ..., K	ENTER↑ [18] leaves α_T mode	→ opens indirect addressing
	Echo	**OP *n* ?** e.g. x≤0?	**OP? *x*** e.g. x≥? Y	**OP? _**	**OP?→ _**
3	User input	Compares x with the real number **0**.	Compares x with the number on stack level **Y**.	*Register number* 0 0 … 9 9, . 0 0 … . 1 5, if the respective register is allocated.	See next page for more about indirect addressing.
	Echo			**OP? *nn*** e.g. x≠? 23	
				Compares x with the number stored in **R23**.	

[18] You may skip this keystroke for register addresses >19 or local registers. The latter start their address with a . – see the section about programming and *Appendix B* below.

1	User input	[RCL], [STO], RCLS, STOS, [aRCL], [aSTO], [VIEW], [VWa+], [x≷], y≷, z≷, t≷, [DSE], [ISG], DSL, DSZ, ISE, ISZ, [ALL], [FIX], [SCI], [ENG], DISP, BASE, bit or flag commands, etc.		
	Echo	OP _ (with α_T mode set), e.g. RCL _[19]		
2	User input	*Stack level or lettered register*[20] [X], [Y], [Z], … , [K]	*Register or flag **number** or number of bit(s) or decimals* (see below for valid ranges)	[→] opens indirect addressing
	Echo	OP *x* e.g. SF K	OP *nn* e.g. SCI 10	OP→ _
3	User input	Sets flag 111.	*Stack level or lettered register* [X], [Y], [Z], … , [K]	*Register number* [0][0] … [9][9], [.][0][0] … [.][1][5], if the respective register is allocated.
	Echo		OP→ *x* e.g. VIEW→L	OP→ *nn* e.g. STO→45
			Shows the content of the register where **L** is pointing to.	Stores *x* into the location where **R45** is pointing to.

Type	Number range[21] (some more registers and flags carry letters)
Registers	0 … 99 for direct addressing of global numbered registers .0 … .15 for direct addressing of local registers 0 … 255 for indirect addressing (≤111 without local registers) } upper limits depend on allocation
Flags	0 … 99 for direct addressing of global numbered flags .0 … .15 for direct addressing of local flags if allocated 0 … 127 for indirect addressing (≤111 without local flags)
Decimals	0 … 11
Integer bases	2 … 16
Bits	0 … 63, word sizes up to 64 bits

[19] For [RCL] and [STO], any of [+], [−], [×], [/], [▲], or [▼] may precede step 2, except in [RCL] [MODE] and [STO] [MODE]. Entering the operator twice will remove it, e.g. [STO][/][/] equals [STO].
Note [ENG] [ENTER↑] calls ENGOVR and [SCI] [ENTER↑] calls SCIOVR. See the index of operations.

[20] Exceptions: [ALL], [FIX], [SCI], [ENG], DISP, and BASE accept lettered registers in indirect addressing only. Else, specifying register **X** as well as RCL T, STO T, RCL× T, STO× T, RCL Z, STO Z, RCL+ Z and STO+ Z require an [ENTER↑] heading the letter, e.g. [STO] [+] [ENTER↑] [Z] for the latter.

[21] For short numbers, you may key in e.g. [5] [ENTER↑] instead of [0][5].

Real Matrices and Vectors

Numbers arranged in table-like grids are called *matrices* by mathematicians. If you do not know of matrices yet, feel free to set them aside – you can use your *WP 34S* perfectly without them.

If you know of matrices, however, note your *WP 34S* features a set of operations for adding, multiplying, inverting and transposing matrices, as well as for manipulating rows in such matrices. In general, the respective commands are building blocks designed to provide the low level support routines for creating more useful matrix functions in the form of keystroke programs. I.e. they represent the basic linear algebra subprograms of the *WP 34S* matrix support. On the other hand, your *WP 34S* also provides functions for computing determinants or for solving systems of linear equations.

A matrix is represented within your *WP 34S* by its *descriptor*, formatted `bb.rrcc` with

rr being the number of rows and

cc the number of columns it features. Thus this matrix has ***rr*** × ***cc*** elements.

These elements are stored in consecutive registers starting at base address |***bb***| .

Example: A descriptor 7.0203 represents a 2×3 matrix – let us call it (*M*). As you know, its six elements are arranged in two rows and three columns, and they are numbered as follows:

$$(M) = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{pmatrix}$$

The matrix descriptor **7**.0203 tells you where the values of these elements are stored:

$$(M) = \begin{pmatrix} \boldsymbol{r07} & \boldsymbol{r08} & \boldsymbol{r09} \\ \boldsymbol{r10} & \boldsymbol{r11} & \boldsymbol{r12} \end{pmatrix}$$

Depending on the current contents of these registers, the actual matrix may look like this:

$$(M) = \begin{pmatrix} -2.3 & 0 & 7.1 \\ 0.4 & 8.5 & -6.9 \end{pmatrix}$$, for example.

If ***cc*** is omitted in a descriptor it defaults to ***rr*** so a square matrix is assumed. For **example**, a descriptor 13.0400 belongs to a 4×4 matrix with its elements stored in **R13** through **R28**. The maximum number of matrix elements is 100 – it is the number of general purpose registers available for such a task.

See the *IOP* and the *catalog MATRIX* for all commands featured.

ATTENTION: Your *WP 34S* cannot know whether a particular real number is a matrix descriptor or a plain number. It is your task to take care of that.

A *vector* may be regarded as a special case of a matrix featuring either one row or one column only. Thus, a vector descriptor looks like `bb.01cc` or `bb.rr01`. Library routines are readily provided for 3D vector calculus.

If you just want to do vector operations in 2D, there are simple alternatives (known for long from earlier calculators) to full-fledged descriptor controlled computations: enter the Cartesian components of each vector in **X** and **Y** (e.g. by converting its polar components into Cartesian ones by [→R], if necessary) and choose one of the following alternative opportunities:

1. use [Σ+] for additions or [Σ−] for subtractions and recall the result via SUM, or
2. calculate in *complex domain* (see next paragraph). Therein, vector multiplication is possible, too, using the commands ^{C}DOT or ^{C}CROSS.

Turn to a good textbook covering *linear algebra* for more information.

Complex Calculations

Mathematicians know of more complicated items than real numbers. There are also *complex numbers*. If you do not know of them, leave them aside – you can use your *WP 34S* perfectly without them.

If you know of complex numbers, however, note your *WP 34S* supports many operations in complex domain as well. The key [CPX] is employed as a prefix for calling complex functions. E.g. [CPX] [f] [COS] calls the complex cosine, and it is displayed and listed as ^{C}COS (the elevated C is the signature for complex functions in your *WP 34S*).

> **ATTENTION 1: All functions operating on complex numbers require them in <u>Cartesian coordinates exclusively</u> on your *WP 34S*. Each such number takes two adjacent registers[22]: the lower one for its real part and the higher one for its imaginary part.**
>
> You may use [→P] to convert $x_c = x + i \cdot y$ to its polar equivalent $x_c = r \cdot e^{i\varphi}$ any time you like – just remember reverting this by [R←] before starting a complex calculation.
>
> **ATTENTION 2: All complex functions work with <u>radians exclusively</u> where an angular input is required, and return radians where they output angular data.** As a reminder, **RAD** is lit when [CPX] is pressed. The user is responsible for providing suitable input values; there will be <u>no</u> automatic conversion of angular data if the angular mode is switched.

Generally, if an arbitrary <u>real</u> function **f** operates on …

- … one real number x only, then its complex sibling ^{C}**f** will operate on the complex number $x_c = x + i \cdot y$. Note x_c will consume two stack levels.
- … one register, e.g. **R12**, then ^{C}**f** will operate on two registers, e.g. **R12** and **R13**.
- … x and y, then ^{C}**f** will operate on x, y, z and t.

[22] The *HP-42S* supported a special data type for complex numbers. This is not viable using the hardware of the *HP-30b*, however. See *Appendix F* for the reasons.

Where **one-number** real functions replace x by the result $\mathbf{f}(x)$, one-argument complex functions replace x by the real part and y by the imaginary part of the complex result $^{C}\mathbf{f}(x_c)$. Higher stack levels remain unchanged. Such functions are e.g. $^{C}1/x$, ^{C}ABS, ^{C}FP, ^{C}IP, ^{C}ROUND, $^{C}x!$, $^{C}x^2$, $^{C}\sqrt{x}$, ^{C}**+/−**, $^{C}\Gamma$, the logarithmic and exponential functions with bases 10, 2, and e, as well as hyperbolic and trigonometric functions and their inverses.

Two-number real functions replace x by the result $\mathbf{f}(x, y)$ as shown above. In complete analogy, two-argument complex functions replace x by the real part and y by the imaginary part of the complex result $^{C}\mathbf{f}(x_c, y_c)$. The next stack levels are filled with the complex contents of higher levels, and the complex number contained in the top two stack levels is repeated as shown overleaf. Such complex functions are the basic arithmetic operations in complex domain as well as ^{C}IDIV, $^{C}\mathrm{LOG}_x$, $^{C}y^x$, $^{Cx}\sqrt{y}$, $^{C}\beta$, and $^{C}||$. Turn to the stack diagrams overleaf for further details.

Where a complex operation (such as ^{C}RCL) does not consume any stack input at all but just returns a complex number, this number will be pushed on the stack taking two levels.

Whenever a complex result is displayed, a capital **C** is lit in the upper left corner of the *LCD* reminding you to look also at stack level **Y** at least.

Calculating with complex numbers uses two registers or stack levels for each such number as explained above and shown here:

	Level address	Assumed stack contents at the beginning:	Stack contents after executing … … the complex stack register operations: ^{C}ENTER	^{C}FILL	^{C}DROP	^{C}x⇆y	^{C}R↓	^{C}R↑	^{C}LASTx	… complex functions of: … one number such as $^{C}x^2$	… two numbers such as ^{C}/
With **4** stack levels	T	Im(y_c) = Im(t_c)	Im(x_c)		$y_c = t_c$	Im(x_c)	x_c		x_c	$y_c = t_c$	$y_c = t_c$
	Z	Re(y_c) = Re(t_c)	Re(x_c)			Re(x_c)					
	Y	Im(x_c)	Im(x_c)		y_c	Im(y_c)	y_c		*last* x_c	Im($(x_c)^2$)	Im(y_c / x_c)
	X	Re(x_c)	Re(x_c)			Re(y_c)				Re($(x_c)^2$)	Re(y_c / x_c)
With **8** stack levels	D	Im(t_c)	z_c	x_c	t_c	t_c	x_c	z_c	z_c	t_c	t_c
	C	Re(t_c)									
	B	Im(z_c)	y_c	x_c	t_c	z_c	t_c	y_c	y_c	z_c	t_c
	A	Re(z_c)									
	T	Im(y_c)	x_c	x_c	z_c	x_c	z_c	x_c	x_c	y_c	z_c
	Z	Re(y_c)									
	Y	Im(x_c)	x_c	x_c	y_c	y_c	y_c	t_c	*last* x_c	$(x_c)^2$	y_c / x_c
	X	Re(x_c)									

So, an 8-level stack gives you the same flexibility in complex domain you are used to with a 4-level stack in real domain. See the *IOP* for all commands supported in complex domain. Many of them are contained in the *complex X.FCN catalog*.

You can use complex domain for 2D vector algebra as well. The functions ^{C}ABS, ^{C}+, ^{C}–, ^{C}CROSS, ^{C}DOT, and ^{C}SIGN wait for you.

After pressing [CPX], your *WP 34S* allows for the complex operations shown on this *virtual keyboard* (but STOPW represents an application in real domain, see p. 145).

Note that a second [CPX], a [←], or [EXIT] directly after [CPX] will just cancel the prefix [CPX], so you return to the default keyboard assignments.

Constants in complex domain and such calculations occupy two registers like all other complex numbers!

Complex constants are simply entered as ***imaginary_part*** [ENTER↑] ***real_part*** .

Pure real constants, identified by a zero imaginary part, are easily put in for complex calculations as follows:

[0] [ENTER↑] ***real_constant*** , or alternatively

[CPX] ***n*** for integers $0 < n \leq 9$ only.

In programming, [CPX] [h] [CONST] # ***n*** with $0 \leq n \leq 256$ will save steps.

Example: The real number π may be loaded via [0] [ENTER↑] [π] or [CPX] [π] – both will result in $y = 0$ and $x = \pi$.

Pure imaginary constants, on the other hand, having a zero real part, are entered thus:

imaginary_constant [ENTER↑] [0] .

Example: The complex unit i may be loaded via [1] [ENTER↑] [0] or [CPX] [.] – both will result in $y = 1$ and $x = 0$.

Compare the stack mechanics shown on previous page.

Addressing Complex Numbers

1	User input	CPX x=? or x≠?			
	Echo	OP _ (with α_T mode set) e.g. ˹x=_?			
2	User input	0 or 1	*Stack level or lettered register* Z, A, C, L, or J	ENTER↑ [23] leaves α_T mode	→ opens indirect addressing.
	Echo	OP *n* ? e.g. ˹x=0?	OP? *x* e.g. ˹x≠? Z	OP? _	OP?→_
3	User input	Compares $x + i\,y$ with the real number **0**.	Compares $x + i\,y$ with $z + i\,t$.	*Register number* 00 … 98, .00 … .14, if the respective register is allocated	See next page for more about indirect addressing.
	Echo			OP? *nn* e.g. ˹x≠? 26	
				Compares $x + i\,y$ with $r26 + i\,r27$.	

[23] You may skip this keystroke for register numbers >19 or local registers. The latter start with a . – see the section about programming and *Appendix B* below.

1	User input Echo	[CPX] [RCL], [STO], or [x⇄] **OP _** (with α_T mode set) e.g. ᶠRCL _[24]		
2	User input Echo	*Stack level or lettered register* [Z][25], [A], [C], [L], or [J] **OP *x*** e.g. ᶠRCL L	*Register number* [0][0] … [9][8], [.][0][0] … [.][1][4], if the respective register is allocated. **OP *nn*** e.g. ᶠSTO 18	[→] opens indirect addressing, generally working as *in real domain*. **OP→_**
3	User input Echo	This is ᶜLASTx – the real part is recalled from register **L** to **X**, the imaginary part from **I** to **Y**.	*Stack level or lettered register* [X], [Y], … , [K] **OP→ *x*** e.g. ᶠx⇆→Z	*Register number* [0][0] … [9][9], [.][0][0] … [.][1][5], if the respective register is allocated. **OP→ *nn*** e.g. ᶠSTO→45
		Swaps x and the content of the register where z is pointing to, and y with the content of the next one. Assume e.g. $x = 4$, $y = 3$, $z = 20$, $r20 = 5$, $r21 = 6$, then you will find $x = 5$, $y = 6$, $r20 = 4$, $r21 = 3$ after ᶜx⇆→Z.		Stores $x + i\,y$ into 2 consecutive registers, starting with the register $r45$ is pointing to.

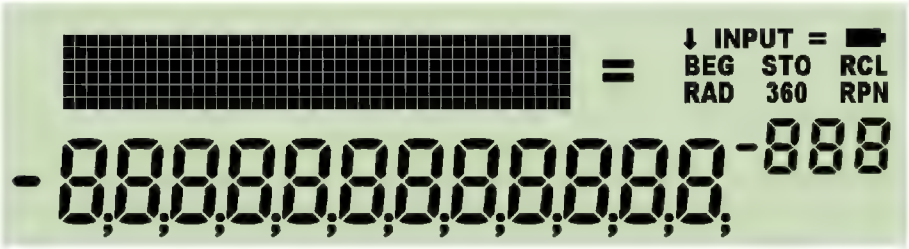
ATTENTION: A complex operation will always affect a pair of registers: the one specified and the one following this. To avoid ambiguity we strongly recommend storing complex numbers with their real parts at even register addresses always. This is guaranteed if you specify an even register number in direct addressing.

[24] For [RCL] and [STO], any of [+], [−], [×], or [/] may precede step 2. See the index of operations.
[25] Exceptions: ᶜRCL Z, ᶜRCL+ Z, ᶜSTO Z, and ᶜSTO+ Z require an [ENTER↑] preceding [Z], e.g. [CPX] [STO] [+] [ENTER↑] [Z] for the latter.

YOUR *WP 34S* IN VARIOUS OPERATING MODES

The Display

The *LCD* is your window to your *WP 34S* – there you see what is going on and what the results are. This display sports a total of 400 elements in three sections: numeric, dot matrix and fixed symbols. The numeric section features a minus sign and 12 digits for the mantissa, as well as a minus sign and 3 digits for the exponent. The dot matrix is 6 dots high and 43 dots wide, allowing for some 7 to 12 characters, depending on their widths. The fixed symbols on the top right side (except the big '=') are called *annunciators.*



The ten annunciators are for indicating modes.

The numeric section in the lower part of the *LCD* is used for displaying numbers in different formats, status data, or message parts. See the examples below for more.

The dot matrix section above is used for passing additional information to the user.

> During command input, for example, the dot matrix displays the command chosen until input is completed, i.e. until all required trailing parameters are entered. The prefixes f, g, h, CPX and → are shown until they are resolved (→ goes with g by default). If you pressed any such prefix erroneously, recovery is as easy as follows:
>
> - f f = g g = h h = CPX CPX = → → = NOP
> - CPX ← = CPX → = CPX EXIT = NOP
> - → ← = → CPX = NOP
> - g f = h f = f
> f g = h g = g
> f h = g h = h
>
> In addressing, progress is recorded as explained in the *tables above* in detail. You may edit or cancel such pending operations by ← or EXIT (see p. *119*ff).

If two or more requests compete for display space, the priorities are:

1. error messages as described in *Appendix C*,
2. special information as explained below,
3. information about the modes your *WP 34S* is running in.

Recognizing Calculator Modes

The **annunciators** or specific characters either in the dot matrix or in the exponent section indicate most modes and system states:

Indicator	Set by	Cleared by	Explanation, remarks
↓	[↑]	[↑]	Lower case letters will be entered in alpha mode (see p. 61).
INPUT	[α], αON	[ENTER↑], αOFF, [EXIT]	Alpha mode (see p. 61)
=	*look at right*	*look at right*	Steady if debug mode is turned on (see App. A and H) or while serial I/O is in progress (see App. D). Flashing when the stopwatch is running (except in STOPW). Else clear.
▬	battery low	battery voltage > 2.5 *V*	Your *WP 34S* will shut off when voltage drops below 2.1*V*.
BEG	program pointer at step 000	otherwise	
STO	[P/R]	[P/R], [EXIT]	Programming mode
RCL	flashes while a program is running	program stopped	
RAD	[RAD]	[DEG], [GRAD]	Angular mode (see p. 42)
360	[DEG]	[GRAD], [RAD]	
RPN	almost every command	a *temporary message*	See p. 38 for handling of *temporary messages* in general.
b_ _	[2]	any other BASE setting [a b/c], [d/c], and FRACT will set fraction mode (called FRC). [ALL], [FIX], [SCI], [ENG], [H.MS], [→][H.MS], [H.d], and TIME will set default floating point decimal mode (i.e. DECM).	Binary integer mode
3_ _ ... 7_ _	BASE 3 ... BASE 7		Respective integer mode (see p. 53 for all of these modes)
o_ _	[8]		Octal integer mode
9_ _	BASE 9		Integer mode of base 9
d_ _	[10]		Decimal integer mode
-1_ _ ... -5_ _	BASE 11 ... BASE 15		Respective integer mode
h_ _	[16]		Hexadecimal integer mode
_c_	carry set, [SF][C]	[CF][C], carry clear	Indicate the respective bits set in integer modes (see p. 53).
_ _o	overflow, [SF][B]	[CF][B]	

The indicators following below are all lit in the dot matrix exclusively:

Indicator	Set by	Cleared by	Explanation, remarks
ᶜ	CPX	see p. 35	Transient signal of prefix pending. RAD will be lit together with ᶜ.
C	complex result	else	See p. 29.
D	DBLON	DBLOFF	See Appendix H.
ᶠ	f	see p. 35	Transient signal of prefix pending
ᵍ	g		
G	GRAD	DEG, RAD	Angular mode (see p. 42)
ᵍ →	→	see p. 35	Transient signal of prefix pending
ʰ	h		
M.DY	M.DY, SETUS	any other date or region setting	Date modes (see p. 39)
Y.MD	Y.MD, SETJPN, SETCHN		
÷	entering a catalog	leaving it	See p. 125ff for more about catalogs.

Defaults D.MY and DECM are not indicated. Radix marks and separators are seen in the numeric output immediately, time modes (12h / 24h) in the time string. The numeric format of fraction mode (FRC) is unambiguous as well. Check the examples shown on p. 40ff.

Some mode and display settings may be saved and recovered collectively by STOM and RCLM. These are stack depth, display contrast, complete decimal display settings, trig mode, choices for date and time display, parameters of integer and fraction mode, curve fit model, rounding mode, and precision selected. STOM stores this information in the register you specify. RCLM recalls such a register content and sets the *WP 34S* modes accordingly.

> **ATTENTION:** Ensure that you actually recall mode data – else your *WP 34S* may be driven into very strange settings and it may cost you considerable effort to recover from that unless you find your previous modes stored elsewhere! See Appendix H.

All keyboard input will be interpreted when the input is completed, according to the modes set at input time.

Common Commands Returning Specific Displays: STATUS, VERS, ERR, etc.

Some common commands use the *LCD* in a special way. The respective operations are listed below. Three of them (and more mentioned further below) present ***temporary messages*** as defined here:

> Whenever anything different from the actual contents of **X** in current mode is displayed or any additional information is shown in the dot matrix, these extra data are considered being a *temporary message*. This is further indicated by the annunciator **RPN** turned off as mentioned on p. 36.
>
> If such extra data are displayed outside of a catalog or browser, they will vanish with the next keystroke. Pressing [EXIT] or [←] will just clear the *temporary message* returning to the normal display, any other key will be executed in addition.

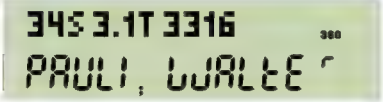
Now here are the common commands delivering special information:

1. [STATUS] returns very useful information about current memory allocation and the space available. It continues showing the user flags set. See p. 127 for a detailed description of this browser.

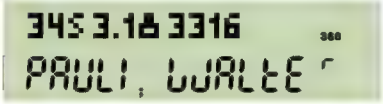
2. VERS generates a *temporary message* showing you version and build of the firmware running on your *WP 34S*.

   - If the basic firmware is installed, the build number will follow the version number immediately.

   - If the timer firmware is installed, a '**T**' will trail the version number.

   - If the firmware for printing is installed, a printer character will be displayed instead.

   See in *Appendix A* how to install these different firmware packs.

3. ERR and MSG display the corresponding error message as a *temporary message*. See *Appendix C* for more.

4. A few far-reaching commands (such as CLALL, for example) will ask you for confirmation before executing. Answer [Y] (by pressing [R/S]) or [N] (pressing [8]). Also [EXIT] or [←] will be read as [N], all other input will be ignored.

Further commands returning special displays and *temporary messages* are specific to particular modes of your *WP 34S* and are thus covered in the sections following.

Floating Point Modes – 1: Introduction and Localisation

Floating point modes cover the 'usual' numbers you calculate with: decimal real or complex numbers, fractions, measured values, times and dates. DECM is the startup default mode of your *WP 34S*. Many commands apply exclusively to it and its sub-modes.

Set display preferences according to your region's practices at once using single dedicated commands:

Command	Radix mark[26]	Time	Date[27]	JG[28]	Three digit separators	Remarks
SETCHN	RDX.	24h	Y.MD	1949	E3OFF	Would require separators every four digits.
SETEUR	RDX,	24h	D.MY	1582	E3ON	SETEUR applies to South America as well. It also applies to Russia, Vietnam, Indonesia, and South Africa, but with deviating JG's.
SETIND	RDX.	24h	D.MY	1752	E3OFF	Would require separators every two digits over 10^5. Applies also to Pakistan, Bangladesh, and Sri Lanka.
SETJPN	RDX.	24h	Y.MD	1873	E3ON	
SETUK	RDX.	12h	D.MY	1752	E3ON	Applies also to Australia and New Zealand. 24h is taking over in the UK.
SETUSA	RDX.	12h	M.DY	1752	E3ON	

[26] See http://upload.wikimedia.org/wikipedia/commons/a/a8/DecimalSeparator.svg for a world map of radix mark use. Looks like an even score in this matter. Thus, the international standard ISO 31-0 allows either a decimal point or a comma as radix mark, and requires a narrow blank as separator of digit groups to avoid misunderstandings. The numeric display hardware of HP-20b or HP-30b, however, does not allow for narrow blanks.

[27] See http://upload.wikimedia.org/wikipedia/commons/0/05/Date_format_by_country.svg for a world map of date formats used. The international standard ISO 8601:2004 states 24h for times, Y.MD for dates. This combination is commonly used in East Asia.

[28] This column states the year the Gregorian Calendar was introduced in the particular region, typically replacing the Julian Calendar (in East Asia, national calendars were replaced in the respective years). Your *WP 34S* supports both 1582 and 1752. See the index of operations for JG1582 and JG1752.

Floating Point Modes – 2: Displaying Decimal Numbers, Fractions, Times, etc.

You find nearly all functions for floating point format control allocated to the top two rows of keys on your *WP 34S*.

1. For **floating point decimal numbers**, startup default allows displaying all digits as long as they fit the display width; it will switch to SCIentific (i.e. mantissa plus exponent) notation to avoid flowing over the display limits. This format is ALL 00, SCIOVR. Besides ALL and SCI, there are two more numeric display formats, FIX and ENG. Their effects can be most easily demonstrated and distinguished using an **example**:

Input \ Format	ALL 00	FIX 4	SCI 4	ENG 4
107.12345678	107.12345678	107.1235	1.0712 2	107.12 0
[1/x] 2 [×]	18.67004725 31 -2	0.0187	1.8670 -2	18.670 -3

Within FIX, the radix mark will always stay at the FIXed position defined. The radix mark floats in the other notations, where e.g. 1.0712 2 represents $1.0712 \cdot \underline{10}^2$, while 18.670 -3 V would mean $18.67 \cdot 10^{-3}\ V = 18.67\ mV$. Within ENG, the exponent will always be a multiple of three corresponding to the unit prefixes in *SI* – thus it is called the ENGineer's notation.

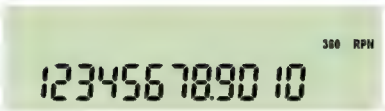
As soon as a number entry is completed, the mantissa will be displayed adjusted to the right, the exponent to the left. Within the mantissa, either points or commas may be selected as radix marks, and additional marks may be chosen to separate thousands.

Assume the display set to FIX 4 again. Key in 12345678 [.] 901 [ENTER↑], and you get

or

with E3ON [29].

Without these separators (i.e. E3OFF), the same number will look like this:



or

[29] These separators may also be beneficial in fraction mode described below.

With ENG 3 and after [+/−], you will get

or

When the last operation executed has returned a *complex* result, C is lit in the top left of the *LCD* pointing to the fact that you will find the result of this function in **X** and **Y**.

Floating point decimal numbers within $10^{-383} \leq |x| < 10^{+385}$ may easily be entered directly in default DECM. Within this range, your *WP 34S* calculates with 16 digits. Values less than 10^{-398} are set to zero (see *Appendix B* for the reasons). For results $|x| \geq 10^{+385}$, error 4 or 5 will appear unless flag **D** is set (see *Appendix C*).

2. [◀] and [▶] display the full number x, i.e. all sixteen digits present internally in the mantissa of standard floating point numbers (imagine a radix mark after the first digit), and all digits of the exponent almost like in scientific display format. All this is shown in one *temporary message*, working like the SHOW command in the *HP-42S*.

   For **example**, [π] 3 [y^x] [◀] returns

   reading 3.100 627 668 …· 10^1 .

3. **Fraction mode** works similarly to the one in the *HP-35S* or the *HP-32SII*. In particular, DENMAX sets the maximum allowable denominator (see the *IOP*). Display will look like in the examples below – fractions are adjusted to the left. If the fraction is exactly equal, slightly less, or greater than the floating point number converted, =, Lt, or Gt is indicated in the exponent, respectively. On your *WP 34S*, fraction mode can handle numbers with absolute values less than 100,000 and greater than 0.0001. Maximum denominator is 9999. Underflows and overflows will be displayed in the format set before fraction mode was entered.

Example: Enter the following:	and you will see:	
[MODE] [D] [▼][▼]	÷ DENANY	in catalog MODE.
[XEQ]		initializes.
0 [MODE] [▼][▼][▼]	÷ DENMAX	
[XEQ]		sets maximum precision.
3 [1/x] [d/c]		since 1/3 > 0.333333333333.
[1/x]		since this is exact.

50 [.] 40625 [−]

This is exact as well.

Pressing [x^2] results in

.

Now, enter [a b/c] for converting this into a *proper* fraction.[30] You get

with a little hook left of the first digit shown. This indicates the leading integer number is displayed incompletely – there are at least two digits preceding 47 but no more display space. Press [◀] (see the previous point) to unveil the integer part of this proper fraction as 2247.

Numeric input in fraction mode is straightforward and logically coherent. If you enter two points, the first is interpreted as a space, the second as a fraction mark:

Key in:	and get in proper fraction mode:	
[1] [2] [.] [3] [.] [4] [ENTER↑]	12 3/4	
[1] [.] [2] [ENTER↑]	1 1/5	(decimal input)
[.] [1] [.] [2] [ENTER↑]	1/2	
[.] [1] [2] [ENTER↑]	3/25	(decimal input)
[1] [.] [.] [2] [ENTER↑]	1 0/1	(= 1 $^0/_2$)[31]

Fraction mode is entered via [a b/c], [d/c], or by entering a second radix mark in numeric input. It is left towards floating point decimal mode via [ALL], [FIX], [SCI], [ENG], [H.d], or [H.MS].

4. There are three **angular modes** featured: DEG, RAD, and GRAD.[32] And *degrees* (DEG) may be displayed in decimal numbers as well as in *hours*, *minutes*, *seconds* and hundredth of *seconds* (H.MS). The usual angular conversions are provided:

[30] This is a new feature compared to previous RPN calculators. By the way, 'proper fractions' cover "echte Brüche" (like ¾) and "gemischte Brüche" (like 2 ½) in German.

[31] This display of an integer number tells you unambiguously your *WP 34S* is in proper fraction mode. For comparison, note the *HP-32SII* reads [1] [.] [.] [2] as ½ – but this is not consistent with its other input interpretations in fraction mode.

Example: Enter the following:	and you will see:	
RAD		Choose *radians* as current angular mode.
π 300 / FIX 4	0.0105	So π/300
→ DEG	0.6000	are exactly 0.6°
→ H.MS	0.3600	or 0°36'0".

The full spectrum of angular conversions provided is seen here:

From	*degrees* H.MS	decimal *degrees*	*radians*	*gon* (*grad*)	current angular mode
… to *degrees* H.MS	—	→ H.MS	—	—	—
… to decimal *degrees*	→ H.d	—	CONV rad→°	CONV G→°	→ DEG
… to *radians*	—	CONV °→rad	—	CONV G→rad	→ RAD
… to *gon* (*grad*)	—	CONV °→G	CONV rad→G	—	→ GRAD
… to current angular mode	—	X.FCN DEG→	X.FCN RAD→	X.FCN GRAD→	—

See the *IOP* for more.

5. **For fitting** measured and accumulated data points with a regression curve, four mathematical models are provided as in the *HP-42S*. See the commands EXPF, LINF, LOGF, and POWERF in the *IOP*. The command BESTF will set your *WP 34S* to select the model resulting in the greatest absolute correlation coefficient (see CORR).

   As shown in this **example**

   ,

   the fit model applied is displayed *temporarily* after each command related to fitting (i.e. after CORR, COV, L.R., s_{XY}, $\hat{x}$, $\hat{y}$). Like with all other auto-functionality, you should know what you are doing here.

[32] This traditional calculator notation is misleading in German at least: DEGrees on your *WP 34S* mean "*Grad*", while GRADs are called "*Gon*".

6. In **H.MS display mode**, entered via [H.MS], decimal numbers are converted and displayed in a format `hhhh°mm'ss.dd"` with the number of *hours* or *degrees* limited to 9,000. This *temporary message* may look like

for RDX. or

for RDX, .

For decimal times less than 5*ms* or 0.005 *angular seconds* but greater than zero, an u for underflow will be lit in the exponent section:

.

Note there are no leading zeroes in the *hours*, *minutes,* and *seconds* sections.

For times or angles exceeding 9,000, an o is shown in the exponent section signaling an overflow, and the value is displayed modulo 9,000.

For **example**:

will become after [H.MS]

until the next key is pressed.

7. **WDAY** returns a display looking like the following for an input of 13.01201 in default D.MY mode (equivalent to inputs of 2010.0113 in Y.MD or 1.13201 in M.DY):

.

Expect similar displays after DAYS+.[33]

[33] Dates before the year 8 may be indicated differently to what they really were due to the inconsistent application of the leap year rule before this. We count on your understanding and hope this shortcoming will not affect too many calculations.

Floating Point Modes – 3: Real Statistical Calculations

Besides the basic functions [%] and [Δ%], you will find a lot of statistical commands embedded in your *WP 34S*, going far beyond the Gaussian distribution. They are all concentrated in the light green frame shown in the picture. Many preprogrammed operations are implemented in your *WP 34S* for the first time ever in an *RPN* calculator – we packed-in everything we always had missed.

The labels below [5] and following to the right cover sample statistics. The five labels at left deal with general probability.

[!], [Cy,x], and [Py,x] stand for $x!$, COMB, and PERM as introduced above.

The shifted functions of [4] cover statistical distributions: standard normal [Φ] and its inverse [Φ⁻¹], as well as the catalog [PROB] covering nine more continuous distributions and three discrete ones. All these functions have a few features in common:

- Discrete statistical distributions (like Poisson and Binomial) are confined to integers. Whenever your *WP 34S* sums up a *probability mass function* (*pmf* [34]) $p(n)$ to get a *cumulated distribution function* (*cdf*) $F(m)$ it starts at $n = 0$. Thus,

$$F(m) = \sum_{n=0}^{m} p(n) = P(m) \ .$$

- Whenever your *WP 34S* integrates a function, it starts at the left end of the integration interval. Thus, integrating a continuous *probability density function* (*pdf*) $f(x)$ to get a *cdf* $F(x)$ typically works as

$$F(x) = \int_{-\infty}^{x} f(\xi) d\xi = P(x) \ .$$

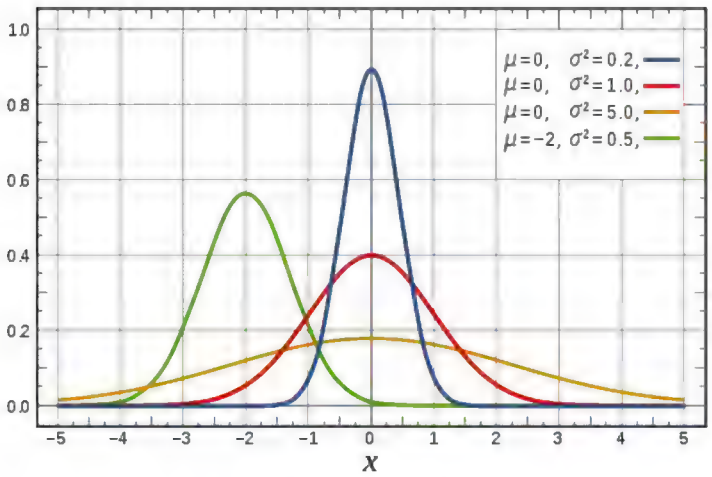
[34] In a nutshell, discrete statistical distributions deal with "events" governed by a known mathematical model. The *pmf* then tells the probability to observe a certain number of such events, e.g. 7. And the *cdf* gives the probability to observe up to 7 such events, but not more.
For doing statistics with continuous statistical variables – e.g. the heights of three-year-old toddlers – similar rules apply: Assume we know the applicable mathematical model. Then the respective *cdf* gives the probability for their heights being less than an arbitrary limit value, for example less than $1m$. And the corresponding *pdf* tells how these heights are distributed in a sample of let's say 1000 children of this age.

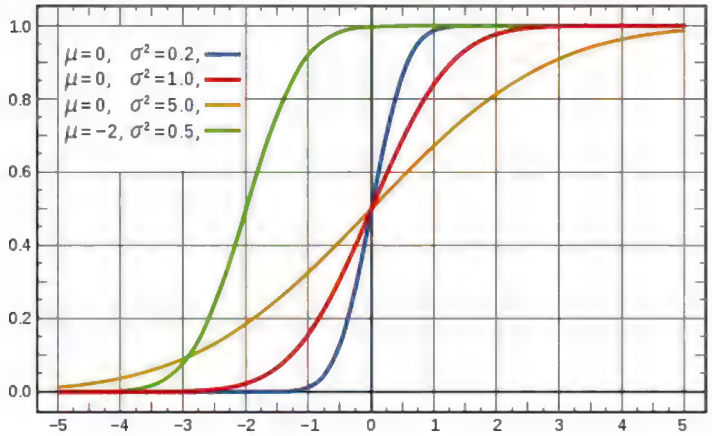
BEWARE: This is a very rudimentary sketch of this topic only – turn to good textbooks about statistics to learn dealing with it properly.

The terms *pmf* and *pdf* translate to German „Dichtefunktion" or „Wahrscheinlichkeitsdichte", *cdf* to „Verteilungsfunktion" or „Wahrscheinlichkeitsverteilung".

- Many frequently used *pdf*s and *pmf*s look more or less like the ones plotted in the upper diagram here. The corresponding *cdf*s are shown below, using the same colors. Typically, any *cdf* starts with a very shallow slope, becomes steeper then, and runs out with a decreasing slope while slowly approaching 100%. This holds even if the respective *pdf* does not look as nice and symmetric as the examples here.

  Obviously you get the most precise results on the left side of the *cdf* using P. On its right side, however, where P approaches 1, the *error probability* $Q = 1 - P$ is more precise. Thus, your *WP 34S* also computes Q for each distribution, independent of P.





- On your *WP 34S*, with an arbitrary *cdf* named ***XYZ*** you will find the name

  XYZ$_u$ for its *error probability* Q (also known as upper tail probability), if applicable,

  XYZ^{-1} for the inverse of the *cdf* (the so-called *quantile function* or *qf*), and

  XYZ$_P$ for its *pdf* or *pmf*.

  This naming convention holds for **Binom**ial, **Cauch**y (a.k.a. *Lorentz* or *Breit-Wigner*), **Expon**ential, *Fisher's* **F**, **Geom**etrical, **LogNorm**al, **Logis**tic, **Norm**al, **Poiss**on, *Student's* **t**, and **Weibu**ll distributions. Chisquare and Standard Normal (*Gaussian*) distributions are named differently for reasons of tradition. See the *catalog PROB* and the respective entries in the *IOP*. For those interested, some mathematical background and further links may be found in *Appendix I*.

There is also a wealth of commands for sample and population statistics featured, applicable in one or two dimensions. All these are located to the right of key [4] on the keyboard. After clearing the summation registers by [CLΣ], use [Σ+] (or the primary shortcut [Σ+] top left on the keyboard) to accumulate your experimental data – typically counted or measured values – as on the *HP-42S* etc.; weighted data require the weight in **Y**, pairs of data or coordinates of data points must be provided in **X** and **Y** as usual. [Σ−] is provided for easy data correction.

Regarding the analysis functions featured, you find the arithmetic mean [x̄] and standard deviation [s] handy on the keyboard, as well as a forecasting function [ŷ] and the correlation coefficient [r] (see CORR) for four different regression models (linear, exponential, logarithmic, and power – see the commands LINF, EXPF, LOGF, and POWERF). Many more functions (such as standard errors, covariances, means and standard deviations for weighted data, standard deviations for populations, curve fitting parameters, geometric means and scattering factors) are in *STAT*, and all the accumulated data are in *SUMS* – just look them up in these catalogs and then check the respective entries in the *IOP*.

To get an idea of the possibilities provided and of some constraints inherent to statistics, see the two sample applications shown here:

Application 1: Assume you own a little tool shop and want to know the quality of the parts you produce. You drew a *representative sample* of nominally equal parts and precisely measured their real sizes. How can you know your batch will be ok? That is easy, based upon analysis of that sample.

Example: 10 pins drawn from a production batch, diameters measured: 12.356, 12.362, 12.360, 12.364, 12.340, 12.345, 12.342, 12.344, 12.355, and 12.353. – From earlier investigations, it is known that diameters from this production process are *Gaussian*.

Now you must know your objective:

- Do you want to know what pin diameters you will get? Statistics cannot tell you about <u>all</u> of them here but it will tell you where to find <u>almost all</u> (e.g. 99%) of them.

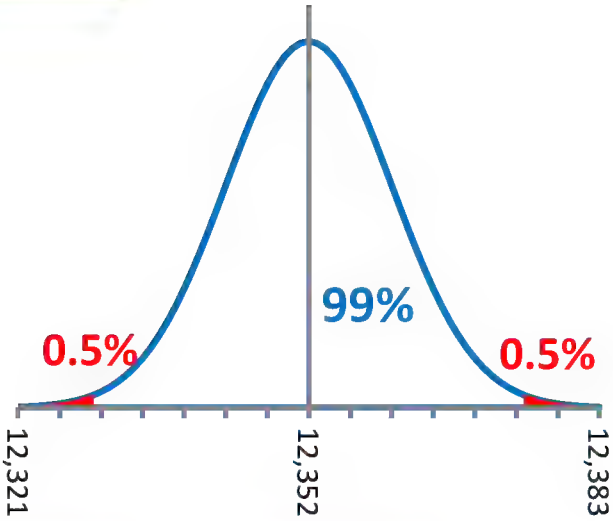
Example (continued): Reset the summation registers and accumulate the ten measured values:

[CLΣ] [FIX] 03	
12.356 [Σ+]	1.000
12.362 [Σ+]	2.000
12.360 [Σ+]	3.000
12.364 [Σ+]	4.000
12.340 [Σ+]	5.000
12.345 [Σ+]	6.000
12.342 [Σ+]	7.000
12.344 [Σ+]	8.000
12.355 [Σ+]	9.000
12.353 [Σ+]	10.000

By knowing these pin diameters are from a *Gaussian* process, you get the best estimates for the mean and standard deviation of your batch by pressing

x̄ STO J	12.352	and
s STO K	0.009	.

We stored these two values in preparation for the next step already.



So based on the ten pin sample analyzed, you may expect 0.5% of all pins with diameters smaller than

0.005 PROB O ▲	⇕ Norml⁻¹	XEQ	12.330

and another 0.5% with diameters greater than

0.995 PROB	⇕ Norml⁻¹	XEQ	12.375 .

- Are you interested in the mean pin diameter of your batch? Do you want to know the limit below which it will lie with a probability of e.g. 95%?[35] Then determine the applicable sample mean value and the size of its variation. Use them to find said limit.

Example (continued): Since it is a *Gaussian* process, the arithmetic mean is applicable, the standard error tells its variation, and *Student's t* is needed. Press …

SUMS	⇕ nΣ	XEQ	10.000
1 − STO J			9.000
STAT S ▼	⇕ SERR	XEQ	0.003
0.95 PROB U ▲	⇕ t⁻¹(p)	XEQ	1.833
x			0.005
x̄			12.352
x⇄y R↓ [36] +			12.357

So the odds are 95% that the mean pin diameter of that batch will be less than the *confidence limit* 12.357. But remember there is an inevitable chance of 100% – 95% = 5% that it will be greater than that.

[35] This value of 95% is called the *confidence level* of this calculation. 99% are frequently applied as well.

[36] x̄ returns two numbers: x̄ and ȳ. Only x̄ is interesting in this example. Pressing x⇄y R↓ moves ȳ quickly out of the way. In a program, x⇄y followed by DROP is a better alternative.

- Do you want both upper and lower limits confining the mean pin diameter of that batch? Take 0.025 and 0.975 as the arguments in two subsequent calculations using the *qf* to get both limits below and above the sample result.

Example (continued, assuming **J** loaded in the step before):

STAT S ▼	⇕ SERR	XEQ	0.003	
0.025 PROB U ▲	⇕ t⁻¹(p)	XEQ	-2.262	
× |x|			0.006	
x̄			12.352	
x≷y R↓ x≷y			0.006	
−			12.346	= lower limit.
RCL L			0.006	= last x
2 × +			12.358	= upper limit.[37]

Now you know you can expect the future mean diameter of that batch between the *confidence limits* 12.346 and 12.358 – with a confidence of 95%. This means here will be a chance of 2.5% that it will be smaller than the lower limit and an equal chance that it will be greater than the upper limit.

These chances are an inevitable consequence of the fact that you know something about a *sample* only (being a limited, usually small number of specimens drawn from a population), but want or have to tell something about said total population.[38] If you cannot live with these chances or the widths of the confidence limits, do not blame statistics but collect more (or more precise) data instead.

Application 2: Assume you have taken a sample out of a process at day 1. Then you have changed the process parameters, waited for stabilization, and now have taken another sample of same size at day 2. Being serious, you have thoroughly measured and recorded the critical value (e.g. a characteristic dimension) for each specimen investigated at both days. Now: do the results of both samples show any *significant difference*? The following simple three-step test is well established. It may easily save yourself some unwanted embarrassments in your next presentation or after your next publication[39] :

[37] The upper limit can be calculated this easy way since $t^{-1}(p)$ is symmetric.

[38] Statisticians call these chances '*probabilities of a type I error*' or '*probabilities of an error of the first kind*'. By the way, '*confidence limit*' translates to German "*Vertrauensbereichsgrenze*", '*confidence level*' to „*Vertrauensniveau*", and '*type I error*' to "*Fehler 1. Art*".

[39] This test goes back to *DGQ* (*Deutsche Gesellschaft für Qualität*). It assumes your data are drawn from a Gaussian process, which is frequently the case in real life (but needs to be checked). Note the term '*significant*' is well defined in statistics – this definition may deviate from common language.
Generally, standard confidence limits and levels, also those defined for indicating *significant differences*, may depend on the country or industry you are working in. Be sure to check the applicable valid standards.

1. Let your *WP 34S* compute $\bar{x}$ and the *standard error* s_E for both samples, then their *normalized distance* $d = \frac{|\bar{x}_1 - \bar{x}_2|}{\sqrt{s_{E1}^2 + s_{E2}^2}}$. Assume you are working with <u>four</u> stack levels still, this calculation could look like the following:

STAT S ▼	⇕ SERR	
XEQ		returns both standard errors in **X** and **Y**.
x^2 x⇄y x^2 + $\sqrt{x}$		this is the complete denominator now.
$\bar{x}$ − |x| x⇄y / STO D		and this is d.

Also provide the *degrees of freedom* for the next two steps:

SUMS	⇕ nΣ	
XEQ		recall the number of specimens measured.
1 −		calculate the *degrees of freedom*
STO J		and store them where they belong for the next step.

2. Let your *WP 34S* calculate the critical limit t_{cr} of *Student's t* for f *degrees of freedom* and a probability of 97.5% now:

.975	0.975	
PROB U ▲	⇕ t-1(p)	as mentioned above, the requested *qf* lives in catalog PROB.
XEQ		executes this function, taking the degrees of freedom stored in **J** to get t_{cr}.

If $d < t_{cr}$ then the test indicates the difference between both samples being due to random deviations only. Congratulations – you have got a robust process regarding the parameters you changed.

3. Let your *WP 34S* compute a new critical limit t_{cs} for f and 99.5%:

.995	0.995	
PROB	⇕ t-1(p)	note this function shows up immediately when opening the catalog again.
XEQ		get t_{cs}.

If $d \geq t_{cs}$ then the test indicates a *significant difference* between both samples. Congratulations – your parameter change caused an effect.

For $t_{cr} \leq d < t_{cs}$, however, you cannot decide based on the information provided – your samples may contain too little data or your measurements were not precise enough or the process is scattering too far – so you better stay silent or mumble something like "investigation in progress".

We strongly recommend you turn to a good statistics textbook for more information about statistical methods in general, the terminology used, and the mathematical models provided.

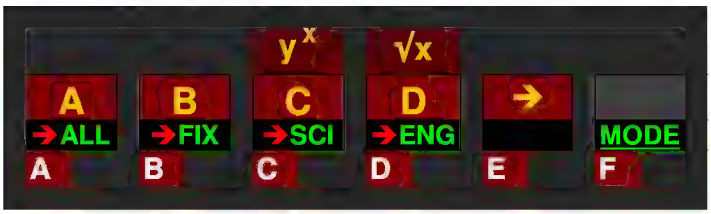
Integer Modes – 1: Introduction and Virtual Keyboard

Integer modes are entered as described on p. 36. They are designed to deal exclusively with integers – in input, output, and calculations. This is useful e.g. for computer logic and system programming – typical applications of an *HP-16C*. Your *WP 34S* contains all the functions of the *HP-16C* and more, and it allows for integer computing in fifteen bases from binary to hexadecimal (see overleaf).

In integer modes, functions such as SIN make no sense for obvious reasons. Thus, for integer bases ≤ 10, the virtual keyboard of your *WP 34S* will look as shown left (where labels headed by a red arrow will leave integer modes when called, typically returning to floating point decimal mode).

For base 16, on the other hand, primary functions of the top six keys will be reassigned automatically, becoming direct numeric input. So this row will then look virtually as shown below.

Wherever a default primary function is not primary anymore after reassignment, prefix f will allow for accessing it (e.g. f D will call √x here[40]). To ease operation, pressing any key (or a sequence of prefix(es) and a key) will display its present assignment in the top line for checking. If the last key is held down for > 0.5 seconds, the display will fall back to NULL and no operation will be executed.



Calculating in bases 11 … 15, those keys not needed for numeric input will work as shown in the first picture above. In any integer base, attempts to enter an illegal digit from the keyboard – such as e.g. 4 in binary – will be blocked.

[40] In such cases, operations printed golden on the key plate cannot be called anymore. This means for the key D, for example, we cannot access TAN in hexadecimal mode – no loss here, of course. Reassignments are generally chosen this way. – Note f D may call a program if defined.

Integer Modes – 2: Displaying Integer Numbers

In integer modes, the mantissa section of numeric display shows the integer in **X**. The exponent section is not needed for numeric display here but may transmit other information. Sign and first digit of the exponent indicate the base set:

Base	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Exponent starts with	b	3	4	5	6	7	o	9	d	-1	-2	-3	-4	-5	h

Carry and overflow – if set – show up as a c in the second or an o in the third digit of the exponent, respectively. So in base 12, the exponent section may well look like -2co . Generally, carry and overflow behave like they do in the *HP-16C*. Note that they are flags – if you want to set, clear, or check them individually, use the flag commands of your *WP 34S.*

Word size and complement setting are indicated in the dot matrix using a format `xx.ww`, with `xx` being **1c** or **2c** for 1's or 2's complement, respectively, **un** for unsigned, or **sm** for sign-and-mantissa mode. Startup default is **2c**. These modes control the handling of negative numbers. They are understood most easily with a little **example**:

Set your *WP 34S* to WSIZE 12, LZON. This setting allows seeing all 12 bits in one *WP 34S* display easily. Enter 147. Then turn to 1COMPL, BASE 2. You will see:

1c 12 1 ... 0000 1001 0011 b and – after [+/-] – 1c 12 1 ... 1111 0110 1100 b.

Forget the 1 top right for the moment – it will be explained later here. Note the low *byte* of our number is displayed larger than its top four bits for easy reading. Obviously [+/-] in **1c** inverts every bit, being equivalent to [NOT] here.

Return to the original number via [+/-] now, choose 2COMPL and you will get:

2c 12 1 ... 0000 1001 0011 b and – after [+/-] – 2c 12 1 ... 1111 0110 1101 b.

Note the negative number equals the inverse plus one in **2c**.

Now return again to the original number via [+/-], choose SIGNMT and you will see:

sm 12 1 ... 0000 1001 0011 b and – after [+/-] – sm 12 1 ... 1000 1001 0011 b.

Negating a number will just flip the top bit in **sm** (hence the name of it).

Finally, return to the original number via [+/−], choose UNSIGN and you will get:

un 12 1
0000 1001 0011 b and – after [+/−] – un 12 1
1111 0110 1101 b o.

Note the second number looks like in **2c**, but in addition an overflow is set here.

This needs explanation, since changing signs has no meaning in **un** per definition[41], where the most significant bit adds magnitude, not sign, so the largest value represented by a 12-*bit* word is 4095 instead of 2047. Thus, [+/−] should be illegal here or result in no operation at least. Nevertheless, [+/−] in unsigned mode was allowed and implemented this way in the *HP-16C*, so we follow this implementation for sake of backward compatibility.

Thus, pressing [+/−] will not suffice anymore for returning to the original number here; you must also clear the overflow flag by [CF] [B] explicitly (see p. 24).

As you have seen, positive numbers stay unchanged in all those four modes. Negative numbers, on the other hand, are represented in different ways. Therefore, taking a negative number in one mode and switching to another one will lead to different interpretations. The fixed bit pattern representing e.g. -147 d in default **2c** will be displayed as -146 d in **1c**, -1901 d in **sm**, and 3949 d in **un**.

Keeping the mode (e.g. **2c** again) and changing the bases will produce different views of the constant bit pattern as well. You will notice that the displays for bases 4, 8, and 16 will look similar to those shown above, presenting all twelve bits to you, while in the other bases a signed mantissa will be displayed instead.

Compare for **example** the outputs

2c 12
331231 4 for base 4 and

2c 12
-1042 5 for base 5.[42]

Let us look to bigger words now: For **example**, turn to UNSIGN, LZOFF, WSIZE 64, BASE 16, and enter 93A14B6. Then your *WP 34S* will display:

un 64
93A 14b6 h with or un 64
93A 14b6 h

[41] This is clearly stated also in the *HP-16C Computer Scientist Owner's Handbook* of April 1982 on page 30. Unfortunately, however, they did not stick to this.

[42] This takes into account that bases 2, 4, 8, and 16 are most convenient for bit and byte manipulations and further close-to-hardware applications. On the other hand, the bases in between will probably gain most interest in dealing with different number representations and calculating therein, where base 10 is the common reference standard.

without separators selected (see SEPON and SEPOFF).

This number will need 28 digits in binary representation, being 1001.0011.1010.0001.0100.1011.0110. Now return to SEPON and choose BASE 2, then the twelve least significant digits will be displayed initially:

They show up together with an indication '''I that there are four display windows in total with the rightmost shown. The least significant *byte* is emphasized as you know it from the example above. Press ◄ and you will get the more significant *bytes* (note the constant 4-*bit* overlap with the previous display here):

The last display shows the four most significant bits of this binary number as the indication I''' confirms.

If leading zeros were turned on (see LZON), there would be eight display windows (corresponding to eight *bytes*) here, with the four 'most significant' *bytes* containing only zeros.

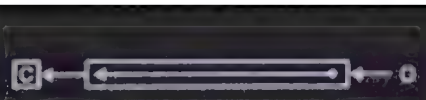
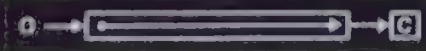
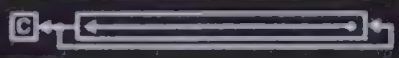
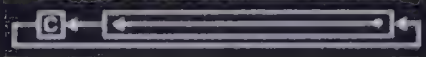
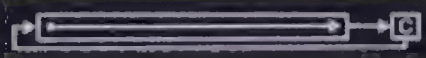
Note numeric input is limited to 12 digits in all integer bases.

Browsing a large integer in steps of eight digits (i.e. byte-wise) is a specialty of binary mode. In any other base the step size is the full display width, i.e. twelve digits without any overlap. See, for **example**, the most and least significant parts of the same number in base 3:

Integer Modes – 3: Bitwise Operations

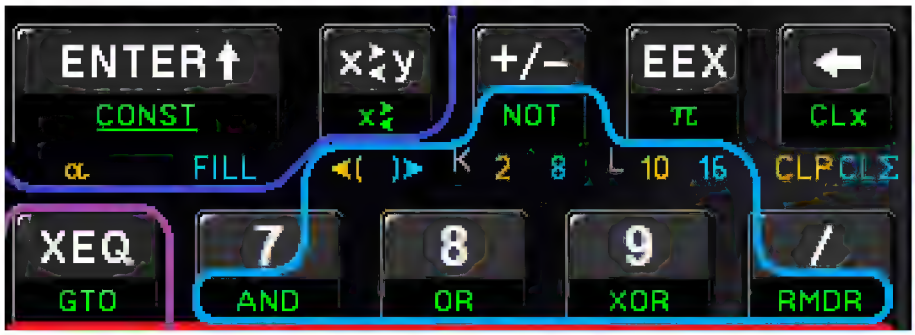
Your *WP 34S* carries all the bitwise operations you may know from the vintage *HP-16C*, plus some more. Generally, bits are counted from right to left, starting with number 1 for the least significant bit. This is important for specifying bit numbers in the operations BC?, BS?, CB, FB, and SB.

The following examples deal with 8-*bit* words showing leading zeros for easy reading. So set WSIZE 8, LZON on your *WP 34S*. For seven functions, you will find the schematic pictures in the table below as they are printed on the backplate of the *HP-16C*. The '**C**' in a box stands for the carry bit there.

			Common start-up *x*
			1011.0011 b
Operation	**Schematic picture**	**Example**	**Output**
Shift Left		SL 1 SL 2	0110.0110 bc 1100.1100 b
Shift Right		SR 1 SR 2 SR 3	0101.1001 bc 0010.1100 bc 0001.0110 b
Arithmetic Shift Right		ASR 3	in 2COMP and 1COMP: 1111.0110 b in UNSIGN: 0001.0110 b in SIGNMT: 1000.0110 b
Rotate Left		RL 1 RL 2	0110.0111 bc 1100.1110 b
Rotate Right		RR 1 RR 2 RR 3	1101.1001 bc 1110.1100 bc 0111.0110 b
Rotate Left through Carry		RLC 1 RLC 2	0110.0110 bc 1100.1101 b
Rotate Right through Carry		RRC 1 RRC 2 RRC 3	0101.1001 bc 1010.1100 bc 1101.0110 b

Note the picture for ASR correctly describes this operation for 1's and 2's complement modes only. In all modes of the *HP-16C*, however, ASR 3 equals a signed division by 2^3, hence the different results for the latter two modes shown above. The other bitwise operations are insensitive to complement mode setting. For further details about those operations, turn to the *IOP*.

Now let us show you the bitwise two-number functions provided as well. Of these, Boole's operators AND, OR, and XOR are found in the light blue frame drawn here:



Again, we will use 8-*bit* words for the following **examples**:

Common input	Y	0110.1011 b
	X	1011.1001 b
Operation		**Output**
AND		0010.1001 b
NAND		1101.0110 b
OR		1111.1011 b
NOR		0000.0100 b
XOR		1101.0010 b
XNOR		0010.1101 b

See the *IOP* for those and further commands working on bit level in integer modes (NOT, LJ and RJ, MASKL and MASKR, MIRROR, RAN#, and nBITS). Unless on the keyboard, the commands mentioned so far are found in the catalog *X.FCN* in integer modes. And there are also BS? and BC? in *TEST*.

Finally, note that no such operation will set an overflow. Carry is only settable by shift or rotate functions as demonstrated above. And ASR is the only bitwise operation being sensitive to complement mode settings – ASR is the link to integer arithmetic operations.

Integer Modes – 4: Integer Arithmetic

Of the four basic arithmetic operations (+, -, ×, and /), the first three work in integer modes as they do in DECM, but with up to 64 digits precision in binary mode. Look at +/- like a multiplication times -1, and at y^x like a repeated multiplication. Depending on the input parameters and the mode settings, the overflow or carry flags will be set in such an operation (see the end of this section).

Divisions, however, must be handled differently in integer modes since the result cannot feature a fractional part here. Generally, the formula $\frac{a}{b} = (a \operatorname{div} b) + \frac{1}{b} \cdot \mathrm{rmdr}(a; b)$ applies, with the horizontal bar denoting <u>real</u> division, div representing <u>integer</u> division, and rmdr standing for the remainder of the latter. While remainders for positive parameters are simply found, remainders for negative dividends or divisors lead to confusion sometimes. The formula above, however, is easily employed for calculating such remainders (also for real numbers – see the first line of this example):

Examples: $\frac{25}{7} = 3 + \frac{1}{7} \cdot 4$ (and for a real case: $\frac{25}{7{,}5} = 3 + \frac{1}{7{,}5} \cdot 2{,}5$)

$$\frac{-25}{7} = -3 + \frac{1}{7} \cdot (-4) \quad \Rightarrow \quad \mathrm{rmdr}(-25; 7) = -4$$

$$\frac{25}{-7} = -3 + \frac{1}{-7} \cdot 4 \quad \Rightarrow \quad \mathrm{rmdr}(25; -7) = 4$$

$$\frac{-25}{-7} = 3 + \frac{1}{-7} \cdot (-4) \quad \Rightarrow \quad \mathrm{rmdr}(-25; -7) = -4$$

In general, $a - b \cdot (a \operatorname{div} b) =: \mathrm{rmdr}(a; b)$.

Unfortunately, there is a second function doing <u>almost</u> the same: it is called mod. With the same pairs of numbers as above, it returns:

$\mathrm{mod}(25; 7) = 4$,

$\mathrm{mod}(-25; 7) = 3$,

$\mathrm{mod}(25; -7) = -3$,

$\mathrm{mod}(-25; -7) = -4$.

So mod returns the same as rmdr if both parameters have equal signs only. The general formula for mod is a bit more sophisticated than the one above:

$a - b \cdot \mathrm{floor}\left(\frac{a}{b}\right) =: \mathrm{mod}(a; b)$ with e.g. $\mathrm{floor}\left(\frac{25}{7}\right) = 3$ and $\mathrm{floor}\left(-\frac{25}{7}\right) = -4$.

By the way, this formula applies to real numbers as well. So it may be used straight-forwardly for calculating e.g. $\mathrm{mod}(25.3; -7.5) = 25.3 - (-7.5) \cdot (-4) = -4.7$.

These four functions are called IDIV, RMDR, MOD and FLOOR in your *WP 34S* for obvious reasons.

Furthermore, the exponential and logarithmic operations, x^2 and $\sqrt{x}$, x^3 and $\sqrt[3]{x}$, COMB and PERM work in integer modes, too. Beyond these keyboard functions and those operations mentioned further above, there are more in the *catalog X.FCN* such as ×MOD and ^MOD. See the *IOP* for further information about them.

Finally, there are conditions where overflow or carry will be touched in integer arithmetic. Note there is a maximum and a minimum integer for each word size and complement setting – let us call them I_{max} and I_{min}. For **example**,

- I_{max} = 15 and I_{min} = 0 for **un 04**, while
- I_{max} = 7 and I_{min} = -8 for **2c 04**,
- I_{max} = 7 and I_{min} = -7 for **1c 04** and **sm 04**.

Now let us start from one incrementing by one and see what will happen:

UNSIGN		2COMP		1COMP		SIGNMT	
0001 b	1 d	0001 b	1 d	0001 b	1 d	0001 b	1 d
0010 b	2 d	0010 b	2 d	0010 b	2 d	0010 b	2 d
...	...	...	...	...	...	...	...
0111 b	7 d	0111 b	7 d	0111 b	7 d	0111 b	7 d
1000 b	8 d	1000 b o	-8 d o	1000 b o	-7 d o	1000 bco	-0 dco
1001 b	9 d	1001 b	-7 d	1001 b	-6 d	0001 b	1 d
...	...	...	...	...	...		
1110 b	14 d	1110 b	-2 d	1110 b	-1 d		
1111 b	15 d	1111 b	-1 d	1111 b	-0 d		
0000 bco	0 dco	0000 bc	0 dc	0001 bc	1 dc		
0001 b	1 d	0001 b	1 d				

From here we start another turn decrementing by one:

UNSIGN		2COMP		1COMP		SIGNMT	
0001 b	1 d	0001 b	1 d	0001 b	1 d	0001 b	1 d
0000 b	0 d	0000 b	0 d	0000 b	0 d	0000 b	0 d
1111 bco	15 dco	1111 bc	-1 dc	1110 bc	-1 dc	1001 bc	-1 dc
1110 b	14 d	1110 b	-2 d	1101 b	-2 d	1010 b	-2 d
...	...	...	...	...	...	...	...
		1001 b	-7 d	1001 b	-6 d	1110 b	-6 d
		1000 b	-8 d	1000 b	-7 d	1111 b	-7 d
		0111 b o	7 d o	0111 b o	7 d o	1000 b o	-0 d o
		0110 b	6 d	0110 b	6 d	1001 b	-1 d

Note the most significant bit is #3 in SIGNMT and #4 in all other modes here. And subtractions are actually additions in SIGNMT.

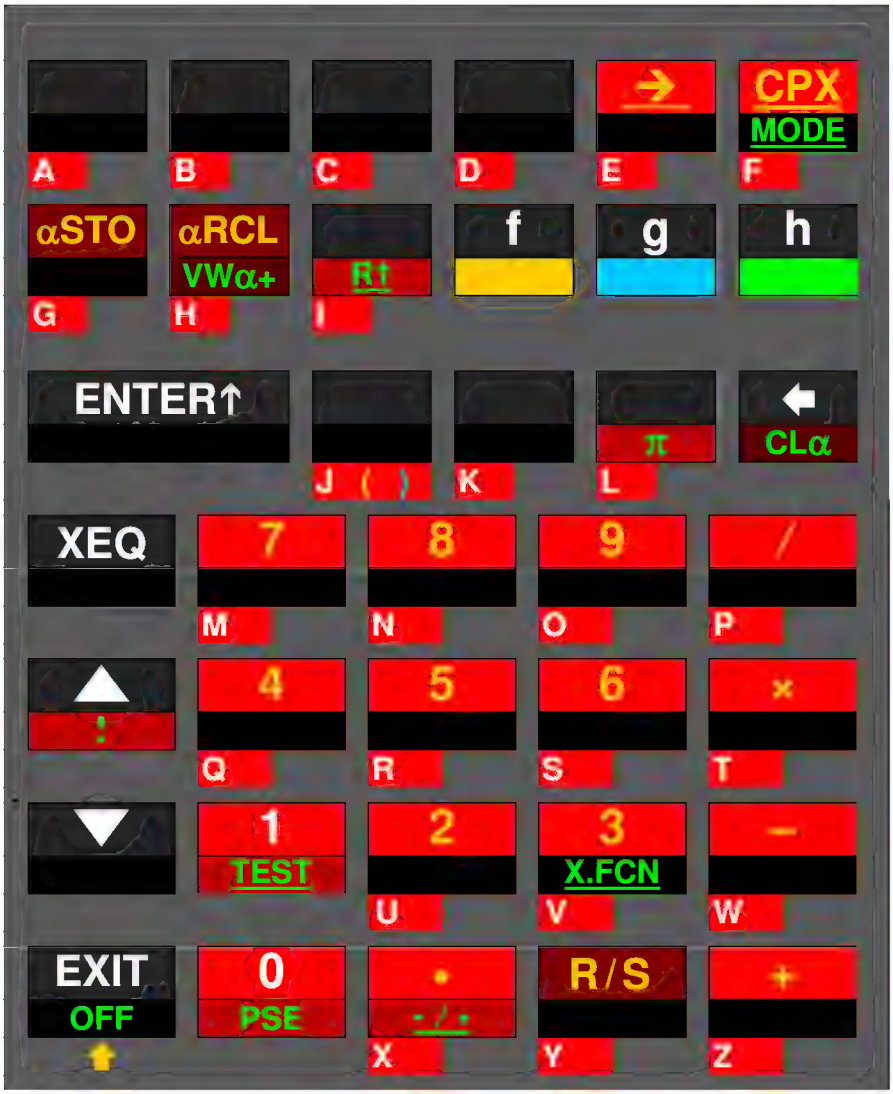
With this, I_{max}, and I_{min}, the general rules for setting and clearing carry and overflow are as follows:

Operation	Effect on carry	Effect on overflow
+, RCL+, STO+, INC etc.	Sets **c** if there is a carry <u>out of</u> the most significant bit, else clears **c**.	Sets **o** if the result would exceed $[I_{max}; I_{min}]$, else clears **o**.
-, RCL-, STO-, DEC etc.	Sets **c** in a subtraction ***m*** – ***s*** if • in 1COMP / 2COMP the binary subtraction causes a borrow[43] <u>into</u> the most significant bit, • in UNSIGN ***m*** is smaller than ***s***, • in SIGNMT ***m*** is smaller than ***s*** <u>and</u> both have the same sign. Else clears **c**.	Sets **o** if the result would exceed $[I_{max}; I_{min}]$, else clears **o**.
×, RCL×, STO×, +/-, $(-1)^x$, x^2, x^3, LCM, $x!$ etc.	None.	Sets **o** if the result would exceed $[I_{max}; I_{min}]$, else clears **o**. Thus in UNSIGN, +/- always sets **o** and $(-1)^x$ does so for odd x.
DBL×	None.	Clears **o**.
/, RCL /, STO /, DBL /, LB, LG, LN, LOG_x, $\sqrt{x}$, $\sqrt[3]{x}$, $\sqrt[x]{y}$	Sets **c** if the remainder is ≠ 0, else clears **c**.	Clears **o** (except setting it for the division $I_{min}/(-1)$ in 2COMP).
2^x	Sets **c** if $x = -1$, or in UNSIGN if $x =$ ***wsize*** or in the other modes if $x =$ ***wsize*** - 1. Else clears **c**.	Sets **o** if the result would become $> I_{max}$, else clears **o**.
y^x, 10^x	Sets **c** for $x < 0$ (and for 0^0), else clears **c**.	
e^x	Sets **c** for $x \neq 0$, else clears **c**.	
ABS	None.	Clears **o** (except setting it for $x = I_{min}$ in 2COMP).

[43] See e.g. the examples above. Both carry and borrow translate to "Übertrag" in German – the 'borrow' in subtraction is a specialty of the USA. See the subtle methodic differences in subtracting shown in detail in http://de.wikipedia.org/wiki/Subtraktion#Schriftliche_Subtraktion . The corresponding English article is far less instructive.

Full Alpha Mode – 1: Introduction and Virtual Keyboard

Alpha mode is typically entered via [α] (see p. 36). It is designed for text entry, e.g. for entering prompts and answers. In this mode, the alpha register is displayed in the upper part of the *LCD*. All direct input goes there, and the numeric line (kept from your last calculation) is accessible by commands only. The display may look like this:



In alpha mode, most mathematical operations are neither necessary nor applicable. So the keyboard is reassigned automatically when you enter alpha mode.

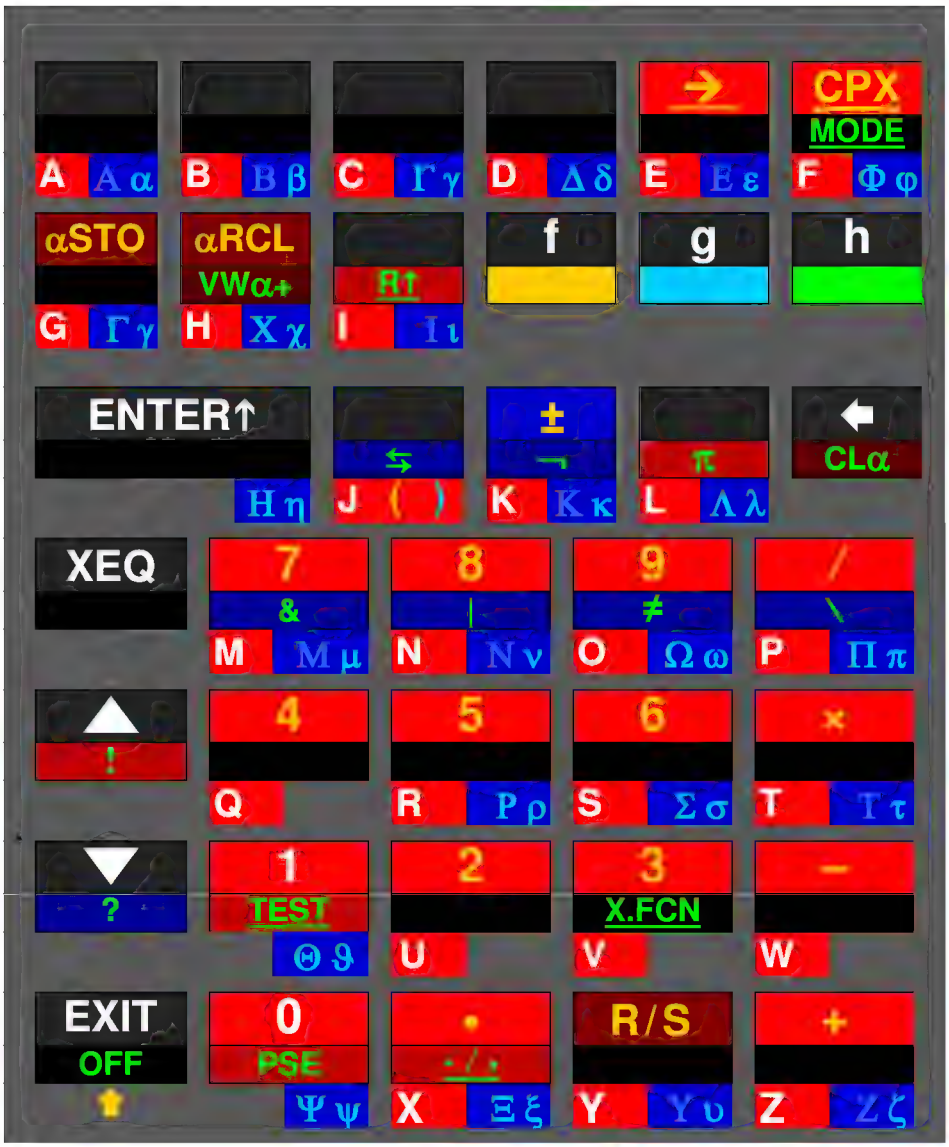
All labels printed here on red background append corresponding characters to ***alpha*** directly or via alpha catalogs. Note four new catalogs become active in this mode. See the keys [→] and [CPX], and the labels [R↑] and [./,] on your *WP 34S*.

Those labels printed on darker red background changed their functionality in other ways. See the keys [STO], [RCL], and [R/S], as well as the labels [VIEW] and [CLx].

Within alpha mode, primary function of most keys becomes appending the letter printed bottom left of this key – grey on the key plate – to ***alpha***. [PSE] appends a space. When ***alpha*** exceeds 30 characters, the leftmost character(s) are discarded. Alpha mode starts with capital letters, and [↑] toggles upper and lower case. As in integer modes, [f] will access default primary functions wherever necessary.[44]

[44] The digits 0 and 1 may also be called using [f][0] or [f][1], respectively.

Looking at the standard labels on the keyboard, we can safely offer you even more in this mode: All labels printed on dark blue background in the virtual keyboard below append characters to ***alpha*** as well. They are related to the labels printed on your *WP 34S* keyboard at these locations, but deviate from them. Prefix g leads to homophonic Greek letters where applicable.[45] And h allows also accessing logic symbols via the Boolean operations.



The alpha catalogs called by f →, f CPX, R↑, ./., and TEST feature even more characters (see p. 132). Check the *IOP* for αSTO, αRCL, VWα+, and more alpha commands.

For a clearer picture of this virtual keyboard – with the extra colors and redundancies removed – turn overleaf.

[45] "Homophonic" according to ancient Greek pronunciation. And we assigned **Gamma** also to **C** due to the alphabet, and **Chi** to **H** since this letter comes next in pronunciation. Three Greek letters require special handling: **Psi** is accessed via g 0 (below PSE), **Theta** via g 1 (below TEST and following **T**), and **Eta** via g ENTER↑. **Omicron** is not featured because it looks exactly like the Latin letter **O** in either case.
Where we printed Greek capitals with lower contrast, they look like the respective Latin letters in our fonts. Professors of Greek, we count on your understanding.

The labels underlined in red call alpha catalogs. Labels underlined in green and labels shown between heavier white key contours do not append characters to ***alpha*** directly – all other labels do.

If you print this virtual keyboard in the size as given below, you can attach it to the back of your *WP 34S* for ready reference. This will serve you well also for learning the Greek letters and their relation to their Latin homophonic siblings. Thereafter, you may slide it 'under the hood' where the batteries live, as sort of a Quick Reference Sheet.

Full Alpha Mode – 2: Displaying Text

Your *WP 34S* features a large and a small alphanumeric font for display. Both are based on fonts by Luiz Viera (Brazil) as distributed in 2004. Some letters were added and some modified for better legibility, also given that the dot matrix of your display is only six pixels high.

See here all characters directly evocable through the virtual alpha keyboard above:

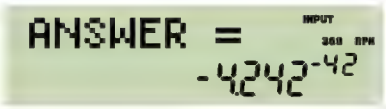
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
αβγδεζηθικλμνξοπρστυφχψω
0123456789 ()+−×/±.!?⇆¬\&|≠

As soon as a string exceeds the visible display using this large font, your *WP 34S* will take the following small font automatically to show as much as possible:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
αβγδεζηθικλμνξοπρστυφχψω
0123456789 ()+−×/±.!?⇆¬\&|≠

Many more characters of both fonts live in the alpha catalogs (see p. 132).

As soon as you enter alpha mode and as long as you stay therein, the contents of the alpha register (abbreviated by ***alpha***) are displayed in the dot matrix, showing its right end (i.e. the last characters it is containing), while the numeric section keeps the result of the last numeric operation. The display may look like:



Note that you may use the big '=' controlled by flag **A** in addition to the dot matrix for message display. Different information may be appended to ***alpha***. See the commands starting with 'α' in the *IOP*.

> For **example**, TIME αTIME allows printing timestamps on messages or creating texts like `It's 11:54:32 PM` or `Um 23:54:32 Uhr` depending on time mode setting and the actual time at execution.
>
> And DATE αDATE will append – depending on date format setting – either `2015-04-16` or `16.04.2015` or `04/16/2015` to ***alpha*** if called on the 7th of April in 2015.

Note ***alpha*** can take up to 30 characters. And your *WP 34S* features a rich set of special letters and further characters, supporting at least 37 languages. So you may easily store an actual Greek message like this, for **example**:

▲ and ▼ will browse such long messages in steps of 6 characters. ▲ will stop with the very first characters shown, ▼ stops showing the right end completely, i.e.

in this very special case.

Having left alpha mode, you can still display ***alpha***: use VIEWα or VWα+ for this – it will show you the left end (i.e. the first characters the alpha register contains).

Nevertheless, do not forget that your *WP 34S* is mainly designed as a programmable calculator.

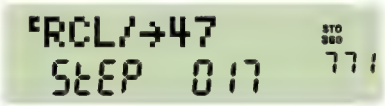
PROGRAMMING YOUR *WP 34S*

Your *WP 34S* is a *keystroke-programmable* calculator. If this statement makes you smile with delight, this section is for you. Else please turn to the *HP-42S Owner's Manual* first for an introduction into keystroke-programming for solving repetitive tasks.

The basic building blocks within program memory are routines (also known as programs). Such routines may contain subroutines, but that is not mandatory. Typically, a routine or subroutine starts with a LBL statement and ends with RTN or END. In between, you may store any sequence of instructions (commands, operations, statements) for repeated use. Choose any operation featured – only a few commands are not programmable. The statements in your routine may use each and every global register provided – there are (almost) no limits. You are the sole and undisputed master of the memory!

This freedom, however, has a price: take care that each routine does not interfere with any other in its quest for data storage space. It is good practice to record the global registers a particular routine uses, and to document their purposes and contents for later reference.

In programming mode (i.e. while editing routines), the numeric display will indicate the current program step (000 – 927) in the mantissa and the number of free steps in the exponent, while the dot matrix will show the command contained in the current step, e.g.:



There is no routine-specific step counting like in the *HP-42S*.

The commands particularly required for programming are located within the violet frame on this keyboard picture. Here you find, starting top left, commands for executing (XEQ) or going to (GTO) a specific program, keys for browsing programs (▲ and ▼), flag commands (SF, CF, and STATUS), catalogs of *TESTs*, programming functions (*P.FCN*), and extra functions (*X.FCN*), commands for SoLVing, integrating, calculating products and sums, keys for EXITing programming mode and for pausing program execution (PSE), some commands for extracting parts (|x|, IP, and FP), keys for

Running / Stopping programs (R/S) and switching between Programming and Run mode (P/R), commands for labeling a routine (LBL) or returning from a routine (RTN), and for loop control (DSE and ISG). See the *IOP* for more about these commands.

Labels

Structuring program memory and jumping around in it is eased by labels you may tag to any program steps – as known from previous programmable pocket calculators. Your *WP 34S* features a full set of alphanumeric program labels as described overleaf. Furthermore, different programs may be separated by END statements. Think of the beginning and the end of program memory containing implicit END statements.

See the next page for addressing labels.

Searching labels, however, obeys the rule below. When a command such as XEQ ***lbl*** is encountered, with ***lbl*** representing a label of one, two or three characters (such as **A**, **BC**, **12**, **Tst**, **Pg3**, **x1µ**, etc.), your *WP 34S* will look for this label using the following method:

1. If ***lbl*** is purely numeric or a hotkey, it will be searched forward from the current position of the program pointer. When an END statement is reached without finding ***lbl***, the quest will continue right after previous END (so the search will stay in the current routine). This is the search procedure for local labels. It is as known from the *HP-41C*.

2. If, however, ***lbl*** is an alpha label of up to three characters of arbitrary case (automatically enclosed in ' like **'Ab1'**), searching will start at program step 000 and cover the entire memory in the order *RAM*, *FM*, and *XROM* (see p. 72 for the latter two), independent of the position of the program pointer. This is the search procedure for global labels.

Tests

Like keystroke-programmable calculators before, your *WP 34S* features a set of tests. Their command names have a trailing '?'. Generally, tests will work as in the *HP-42S*: they will return **true** or **false** in the dot matrix if called from the keyboard; if called in a program, they will execute the next program step only if the test is true, else skip that step. So the general rule reads 'skip if false' (exception: KEY?).

As mentioned above, programs typically end with RTN or END. In running programs, both statements work very similar and show only subtle differences: a RTN statement immediately after a test returning **false** will be skipped – an END will not.

See the *IOP* for more information. All tests are contained in the *catalog TEST*.

Note that there are also commands featuring a trailing '?' but returning numbers (e.g. BASE?) or codes (e.g. KTP?) instead of true or false only – you will find these commands in the *catalog P.FCN*.

Addressing Labels

1	User input	A, B, C, or D	XEQ, GTO, LBL, LBL?, SLV, ∫x/y, π, Σ, αGTO, or αXEQ			
	Echo	**XEQ *label*** e.g. XEQ C	**OP _** e.g. GTO _			
2	User input	Calls the function labeled **C**.	A, B, C, or D	ENTER↑	→ [46]	*2-digit numeric (local) label* 0 0 … 9 9
	Echo		**OP *label*** e.g. Σ B	sets alpha mode. **OP '_**	opens indirect addressing and sets α_T mode. **OP →_**	**OP *nn*** e.g. LBL 07
3	User input		Sums up the function given in a routine labeled **B**.	***Alphanumeric (global) label*** (1 to 3 characters[47])	*Stack level or lettered register* X, Y, Z, … , K	*Register number* 0 0 … 9 9, . 0 0 … . 1 5, if the respective register is allocated[48]
	Echo			**OP '*label'*** e.g. SLV'F1μ'	**OP → *x*** e.g. ∫→T	**OP → *nn*** e.g. XEQ→44
				Solves the function given in the routine labeled **F1μ** (keyed in as explained in footer).	Integrates the function whose label is on stack level **T**.	Executes the routine whose label is in **R44**.

Look up *GTO* in the *IOP* for special cases applying to this command exclusively.

[46] Works with all these operations except LBL .

[47] The 3rd character terminates entry and closes alpha mode – shorter labels need a closing ENTER↑ . For the example given here, press f 2 ENTER↑ CPX 1 f EXIT g 7 and you are done. Statements including alpha labels exceeding one character decrement the number of free program steps by 2.

ATTENTION: LBL A and LBL'A' are different animals! The latter is entered in alpha mode, the first via the hotkey directly.

[48] Some registers may be allocated to special applications. Check the memory table on p. 23.

Local Data

After some time with your *WP 34S* you will have a number of routines stored, so keeping track of their resource requirements may become a challenge. Most modern programming languages take care of this by declaring *local variables*, i.e. memory space allocated from general data memory and accessible for the current routine only – when the routine is finished, the respective memory is released. On your *WP 34S*, *registers* are for data storage – so we offer you *local registers* allocated to your routines exclusively.

> **Example:** Let us assume you write a routine labeled **P1** and need five registers for your computations therein.
>
> Then all you have to do is just enter the command LOCR 5 in **P1** specifying you want five local registers. Thereafter, you can access these registers by using local numbers **.**00 … **.**04 throughout **P1**.
>
> Now, if you call another routine **P2** from **P1**, also **P2** may contain a step LOCR requesting local registers. These will also carry local register numbers **.**00 etc., but the local register **.**00 of **P2** will be different from the local register **.**00 of **P1**, so no interference will occur. As soon as the return statement is executed, the local registers of the corresponding routine are released and the space they took is returned to free memory.

In addition, you get sixteen local flags as soon as you request at least one local register.

Local data holding allows for recursive programs, since every time such a routine is called again it will allocate a new set of local registers and flags being different from the ones it got before.

See the commands LOCR, LOCR?, MEM?, and POPLR in the *IOP* and in *Appendix B* for more information, also about the limitations applying to local data.

Programmed Input and Output, User Interaction and Dialogues

A number of commands may be employed for controlling I/O of programs. In the *IOP*, their behavior is described if they are entered from the keyboard. Executed by a program, however, this will differ in a characteristic way.

When a program is started, the prior display contents are replaced by the Running ProGrAM message and will be updated at certain events only – not after each operation. So where in manual mode each command may change the display immediately with its execution, in automatic mode only PROMPT, PSE, STOP, VIEW, VIEWα, or VWα+ will update the display, and this display will hold until the next command of this kind is encountered.

ERR 0 or MSG 0 are the only ways to get Running ProGrAM back once this message has been replaced by a programmed display.

See the following **examples** – parameters are omitted here:

- Take VIEW, VIEWα, or VWα+ for plain display updates. X is a valid parameter for VIEW and VWα+. Note that frequent updates will slow down program execution, since the anti-flicker logic waits for a complete display refresh cycle before allowing the next update.

- Use one of the following four code segments for displaying messages or other information for a defined minimum time interval, specified by PSE (see the *IOP* for details):

PSE	VIEW PSE	VIEWα PSE	VWα+ PSE
for plain numeric output		for complex alphanumeric messages	

- Ask ('prompt') for numeric input employing one of these four:

STOP	VIEWα STOP	VWα+ STOP	PROMPT	combines VWα+ X and STOP in one command

  Whatever you key in will be in **X** when you continue the program by pressing [R/S] . If you want it elsewhere, take care of it.

- Prompt for alphanumeric input by the following steps:

αON	sets alpha mode and prepares for showing the final part of ***alpha***.
PROMPT	displays this part and waits for user input, terminated by [R/S] .
αOFF	returns to the numeric mode previously set.

  Whatever you key in will be appended to ***alpha*** here. The program will continue when you press [R/S] .

See the *IOP* for more information about these commands and their parameters.

If you press – instead of or after keying in numeric data – one of the hotkeys [A] to [D] in input, the program will call the next routine beginning with a label carrying this name.

The following **example** shows how a typical structure of such a program might look:

```
001  LBL 'MYP'
002   CLα
003   α 'Hel'      Sets up a message …
004   α 'lo!'
005  LBL 00
006   PROMPT       … and stops waiting for user input.
007   GTO 00       [R/S] does nothing, it simply returns to the prompt.
008  LBL A         Called if user input after step 006 was terminated by [A].
009   ENTRY?       Any new numeric data entered by the user?
010   GTO 03       Then go to step 012 where local label 11 lives.
011   XEQ 01       Else call subroutine 01 for computing a new number instead.
012  LBL 03
013   STO 01       Store the new number (input or computed).
014   RTN          Return to prompting via step 007.
```

```
015   LBL 01
...     ...          Compute the new number for missing numeric user input.
...     ...
...     RTN
...   LBL B          Called if user input after step 006 was terminated by [B].
...     ...
...     ...
...     RTN
...   ...
...   END
```

This is the way the TVM application is implemented.

If there is more than one program using labels A to D in *RAM* or *FM*, you must move the program counter (PC) into the desired program and stop there – provided programs are separated by END (see p. 67 to learn which label will be found else).

Keyboard Codes and Direct Keyboard Access

Sometimes, the four hotkeys might not suffice. There is, however, an easy way to extend the number of directly callable subroutines: shorthand addressing of numeric labels using keyboard codes as defined at right. Each key gets a code simply given by its row and column on the keyboard.

A 11	B 12	C 13	D 14	→ 15	CPX 16
STO 21	RCL 22	R↓ 23	f 24	g 25	h 26
ENTER↑ 31		x⇄y 32	+/- 33	EEX 34	← 35
XEQ 41	7 42	8 43	9 44	/ 45	
▲ 51	4 52	5 53	6 54	× 55	
▼ 61	1 62	2 63	3 64	− 65	
EXIT 71	0 72	. 73	R/S 74	+ 75	

Whenever you are asked for the entry of a two-digit label, any of the keys highlighted green in this picture may be used for direct input. The label will then be replaced by the row/column code of the respective key. Keys not available this way (since they have another fixed meaning in this context) may still be used for a short address by pressing f before. Only f itself cannot be used for shorthand addressing.

If, for **example**, you want to link a program to the key [STO], just put label 21 at the beginning of the routine; then it can be called via [XEQ] [STO] by the user conveniently.

The same keyboard codes are returned by the KEY? command, which allows 'real time' response to user input from the keyboard. KEY? takes a register argument (X is allowed but does not lift the stack) and stores the key most recently pressed during program execution in the register specified. R/S and EXIT cannot be queried; they stop program execution immediately. The keyboard is active during program execution – but it is desirable to display a message and suspend the program by PSE while waiting for user input. Since PSE will be terminated by a key press, simply use PSE 99 in a loop to wait for

input. Since KEY? acts as a test as well, a typical user input loop may well look like this **example**:

```
001  LBL 'USR'
002   CLα
003   α 'KEY'      Sets up a message …
004   α ?
005  LBL 00
006   VIEWα        … and displays it.
007   PSE 99       Waits 9.9 s for user input unless a key is pressed.
008   KEY? 00      Test for user input and put the key code in R00.
009   GTO 00       If there was none then go back to step 005.
010   LBL?→00      If a label corresponding to the key code has been defined …
011   XEQ→00       … then call it,
012   GTO 00       … else return to step 005.
```

Instead of the dumb waiting loop, the program can do some computations and update the display before the next call to PSE and KEY? – think of e.g. a lunar landing game.

To be even more versatile, KTP? ***nn*** is designed to return the type of the key pressed if its row / column code is given in register ***nn***. It will return:

- 0 to 9 for the respective digit keys;
- 10 for [.], [+/-], and [EEX];
- 11 for [f], [g], and [h];
- and 12 for the other keys.

An invalid code in the input register will throw an 'Invalid Range' error.

If you decide not to handle the key in your program you may feed it back to the main processing loop of the *WP 34S* with the PUTK ***nn*** command. It will cause the program to halt, and the key will be handled as if pressed after the stop. This is especially useful if you want to allow numeric input while waiting for some special keys like the arrows. This allows writing a vector or matrix editor in user code. After execution of the PUTK command you are responsible for letting the program continue its work by pressing [R/S] or a hotkey.

Flash Memory (*FM*) and *XROM*

In addition to the *RAM* provided, your *WP 34S* allows you to access *FM* for voltage-fail safe storage of user programs and data. The first section of *FM* is a 2*kB* backup region, holding the image of the entire user program memory, registers, and *WP 34S* states as soon as you completed a SAVE. The remaining part of *FM* (up to some 12*kB* depending on setup) will hold programs only. Alphanumeric labels in *FM* can be called via XEQ like in *RAM*. This allows creating program libraries in *FM*. Use CAT to see the global labels already defined – labels in *FM* are tagged with Lib there.

FM is ideal for backups or other long-living data, but should not be used for repeated transient storage like in programmed loops.[49] Conversely, registers and standard user program memory residing in *RAM* are designed for data changing frequently but will not hold data with the batteries removed. So both kinds of memory have advantages and disadvantages you shall take into account for optimum benefit and longevity of your *WP 34S*. Find more about *FM* in *Appendix A*.

Furthermore, there is a memory section called *XROM* (for 'extended *ROM*'), where some command routines live. These, though written in user code, are read-only and thus can be called as commands and executed but not edited. For you, it makes no difference whether a preprogrammed *WP 34S* routine executes in *ROM* or *XROM*. The *XROM* routines, however, are easy to read and understand for users knowing HP keystroke programming – see *http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/xrom/* for the routines present and *Appendix H* for additional information.

[49] *FM* may not survive more than some 10,000 flashes. Thus, we made commands writing to *FM* (like SAVE or PSTO) non-programmable.

INDEX OF OPERATIONS (*IOP*)

All commands available (more than 600) are found below with their *names* and their necessary *keystrokes*. Names printed in **bold** face in this list belong to functions directly accessible on the keyboard; the other commands may be picked from *catalogs*. The command names will show up identically in catalogs and program listings unless specified otherwise explicitly. Sorting in index and catalogs is case insensitive and works in the following order:

␣ 0…9, A…Z, α…ω, ( ) + − × / ± , . ! ? : ; ‘ “ * @ _ ~ → ← ↑ ↓ ↔
< ≤ = ≈ ≠ ≥ > % $ € £ ¥ √ ∫ ∞ & \ ^ | [ ] { } 🖶 #

Superscripts and subscripts are handled like normal characters in sorting. The 🖶 near the end of the sorting order list above is the printer symbol heading all print commands.

Generally, functions and keystroke-programming will work as on the *HP-42S*, bit and integer functions as on the *HP-16C*, unless specified otherwise. There are also functions inspired by other vintage calculators as mentioned in the index. Their manuals may contain helpful additional information about those traditional commands.

A ^{C} heads the names of complex operations (see p. 29). [CPX] is a legal prefix for all functions whose *names are printed in italics* in this list.

Some 300 functions are available on your *WP 34S* for the first time ever on an *RPN* calculator. They got their remarks printed on yellow background. Operations carrying a familiar name but deviating in their functionality from previous *RPN* calculators are marked light red.

The vast majority of remarks for the respective operation start with a number:

(0) represents functions without effects on the stack,

(1) and (2) are for real or complex one- or two-number functions as defined above, and

(3) is for real three-number functions;

(-1) and (-2) stand for functions pushing numbers on the stack thus lifting it by 1 or 2 levels, respectively.

Operations disabling stack lift got a special remark.

Parameters will be taken from the lowest stack level(s) unless mentioned explicitly in the 2nd column of the index – then they must follow the command. Some parameters of statistical distributions must be given in registers **J** and **K** as specified. Some examples of the notation used will be given overleaf.

For the following three examples, assume **R12** contains 15.67 (i.e. ***r12*** = 15.67) generally.

- ***n*** represents an arbitrary integer number which must be keyed in directly, while ***n*** represents such a number which may be specified indirectly via a register as well (as shown in the tables on pp. 26 and 33); and
  n stands for the respective number itself;

  Example: RSD 12 rounds ***x*** to twelve significant digits, while RSD→12 rounds ***x*** to fifteen significant digits.

- ***s*** represents an arbitrary register address which must be keyed in directly, while ***s*** represents such an address which may be specified indirectly as well; and ***s*** stands for the respective contents (i.e. the contents of register ***s*** if specified directly, or the contents of the address where ***s*** is pointing to if specified indirectly);

  Example: STO 12 stores ***x*** into **R12**, while STO→12 stores ***x*** into **R15**.

- ***label*** represents an arbitrary label which must be keyed in directly, while ***label*** represents such a label which may be specified indirectly as well (as shown in the table following p. 67); and
  label stands for the respective label itself.

  Example: GTO 12 goes to local label **12**, while GTO→12 goes to local label **15**.

In the following, each function is listed stating the mode(s) it will work in, indicated by the corresponding annunciators (see p. 36) or abbreviated by their names. In this column, 'integer' stands for an arbitrary integer mode, '&' for a Boolean AND, a comma for an OR, and '¬' for NOT. So e.g. $\mathbf{2^X}$ works in all modes but alpha, even in complex domain. All operations may also be entered in programming mode unless stated otherwise explicitly. Many functions contained in P.FCN make sense in programs only. Red print in the modes column indicates commands working with the quartz crystal and the respective firmware only (see App. A and H).

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
10^x	f 10^x	INPUT	(1)
12h	h MODE 12h	All	(0) Sets 12h time display mode: e.g. 1:23 will become 1:23 AM, 23:45 will become 11:45 PM. This will make a difference in αTIME only.
1COMPL	h MODE 1COMPL	All	(0) Sets 1's complement mode for integers.
1/x	f 1/x	DECM	(1)
	B	DECM	(1) Shortcut working if label B is not defined.
24h	h MODE 24h	All	(0) Sets 24h time display mode. Compare 12h.
2COMPL	h MODE 2COMPL	All	(0) Sets 2's complement mode for integers.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
2^x	f 2^x	¬INPUT	(1)
$\sqrt[3]{x}$	h X.FCN ∛x	¬INPUT	(1)
ABS	f |x|	¬INPUT	(1) Returns the absolute value.
	CPX f |x|	DECM	(1) Returns $r = \sqrt{x^2 + y^2}$ in **X** and clears **Y**.
ACOS	g COS^{-1}	DECM	(1) Returns $arccos(x)$.[50]
ACOSH	g HYP^{-1} COS	DECM	(1) Returns $arcosh(x)$. Note there is no need for pressing f here.
AGM	h X.FCN AGM	DECM	(2) Returns the arithmetic-geometric mean of x and y. See *Appendix I* for more.
ALL	h ALL ***n***	¬INPUT	(0) Sets the numeric display format to show all decimals whenever possible. ALL 0 works almost like ALL in the *HP-42S*. For $x \geq 10^{13}$, however, display will switch to SCI or ENG with the maximum number of digits necessary (see SCIOVR and ENGOVR). The same will happen if $x < 10^{-n}$ and more than 12 digits are required to show x completely. **Example:** Input: 700; ALL 03; 1/x; 10 / — Display: 700; 700.; 0.001428571 43; 1.428571428571^{-4}
AND	h AND	Integer	(2) Works bitwise as in the *HP-16C*. See p. 56.
		DECM	(2) Works like AND in the *HP-28S*, i.e. x and y are interpreted before executing this operation. Zero is 'false', any other real number is 'true'.
ANGLE	h X.FCN ANGLE	DECM	(2) Returns $arctan(y/x)$.
ASIN	g SIN^{-1}	DECM	(1) Returns $arcsin(x)$.[51]
ASINH	g HYP^{-1} SIN	DECM	(1) Returns $arsinh(x)$. Note there is no need for pressing f here.

[50] Precisely, it returns the principal value of it, i.e. a real part ∈ [0, π] in **RAD** or [0, 180] in **360** or [0, 200] in **G**. Compare ISO/IEC 9899.

[51] Precisely, it returns the principal value of it, i.e. a real part ∈ [-π/2, π/2] in **RAD** or [-90, 90] in **360** or [-100, 100] in **G** if flag D is set. Else this result interval becomes e.g. (-π/2, π/2) for the arc tangent. Compare ISO/IEC 9899.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
ASR	h X.FCN ASR *n*	Integer	(1) Works like ***n*** (≤ 63) consecutive ASR commands in the *HP-16C,* corresponding to a division by 2^n . ASR 0 executes as NOP, but loads **L**. See p. 56 for more.
ATAN	g TAN⁻¹	DECM	(1) Returns $arctan(x)$.[51]
ATANH	g HYP⁻¹ TAN	DECM	(1) Returns $artanh(x)$. Note there is no need for pressing f here.
BACK	h P.FCN BACK *n*	¬INPUT	(0) Jumps ***n*** steps backwards (0 ≤ ***n*** ≤ 255). E.g. BACK 1 goes to the previous program step. If BACK attempts to cross an END, an error is thrown. Reaching step 000 stops program execution. Compare SKIP. **ATTENTION:** If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.
BASE	h MODE BASE *n*	All	(0) Sets the base ***n*** for integer calculations, with 2 ≤ ***n*** ≤ 16 (see pp. 52ff for more). Popular bases are directly accessible on the keyboard. Furthermore, BASE 0 sets DECM, and BASE 1 calls FRACT. See there. **ATTENTION:** Stack contents are converted when switching from an integer mode to DECM, and are truncated vice versa. Other register contents stay as they are (see *p.* 159). – BASE 10 is not DECM.
BASE 10	f 10	¬INPUT	
BASE 16	g 16		
BASE 2	f 2		
BASE 8	g 8		
BASE?	h P.FCN BASE?	Integer	(-1) Returns the integer base set.
		DECM	(-1) Returns the integer base set before DECM.
BATT	h X.FCN BATT	DECM	(0) Measures the battery voltage in the range between 1.9V and 3.4V and returns this value.
		Integer	(0) Works like in DECM but returns the value in units of 100mV.
BC?	h TEST BC? *n*	Integer	(0) Tests the specified bit in ***x***.
BestF	h MODE BestF	All	(0) Selects the best curve fit model, maximizing the correlation like BEST does in the *HP-42S*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Binom	h PROB Binom	DECM	(1) Binomial distribution with the number of successes ***g*** in **X**, the probability of a success p_0 in **J** and the sample size ***n*** in **K**. See *Appendix I* for the formula. Binom^{-1} returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, with p_0 in **J** and ***n*** in **K**.
Binom$_P$	h PROB Binomp		
Binom$_u$	h PROB Binomu		
Binom $^{-1}$	h PROB Binom-1		
B_n	h X.FCN Bn	DECM	(1) B_n returns the Bernoulli number for an integer $n > 0$ given in **X**. B_n* works with the old definition instead. See *Appendix I* for details.
B_n*	h X.FCN Bn*		
BS?	h TEST BS? *n*	Integer	(0) Tests the specified bit in ***x*** .
CASE	h P.FCN CASE *s*	INPUT	(0) Works like SKIP below but takes the number of steps to skip from ***s***. **Example**: Assume the following program section: ... 100 CASE 12 101 GTO 01 102 GTO 02 103 GTO 07 104 GTO 05 105 LBL 01 ... 132 LBL 02 ... 153 LBL 05 ... 234 LBL 07 ... In program execution, ***r12*** will be checked in step 100: if $r12 \le 1$ then the program will proceed to step 101 and continue with a jump to step 105, for $r12 = 2$ the program will go to step 102, etc., resulting in a nice controlled dispatcher for $1 \le r12 \le 4$. **ATTENTION 1:** CASE might surprise you for $r12 > 4$ in the example above. Take care of the input you provide in ***s*** ! **ATTENTION 2:** If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.
	h CAT	INPUT	Browser. See p. *125*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Cauch	h PROB Cauch	DECM	(1) Cauchy-Lorentz distribution (also known as Lorentz or Breit-Wigner distribution) with the location x_0 specified in **J** and the shape γ in **K**. See *Appendix I* for details. Cauch^{-1} returns x for a given probability p in **X**, with x_0 in **J** and γ in **K**.
Cauch$_P$	h PROB Cauchp		
Cauch$_u$	h PROB Cauchu		
Cauch $^{-1}$	h PROB Cauch-1		
CB	h X.FCN CB *n*	Integer	(1) Clears the specified bit in x .
CEIL	h X.FCN CEIL	¬INPUT	(1) Returns the smallest integer ≥ x .
CF	g CF *n*	¬INPUT	(0) Clears the flag specified.
CFALL	h P.FCN CFALL	¬INPUT	(0) Clears all user flags.
CLALL	h P.FCN CLALL	¬INPUT	(0) Clears all registers, flags, and programs in *RAM* if confirmed. Not programmable. Compare RESET.
CLP	f CLP	All	(0) Clears the current program, i.e. the one the program pointer is in.
CLPALL	h P.FCN CLPALL	¬INPUT	(0) Clears all programs in *RAM* if confirmed. Not programmable.
CLREGS	h P.FCN CLREGS	¬INPUT	(0) Clears all global and local general purpose registers allocated (see REGS and LOCR). The contents of the stack, **L**, and **I** are kept.
CLSTK	0 g FILL	¬INPUT	Clears all stack registers currently allocated (i.e. **X** through **T** or **X** through **D**, respectively). All other register contents are kept.
	h P.FCN CLSTK		
CLx	h CLx	¬INPUT	Clears register **X** only, disabling stack lift as usual.
CLα	h CLx	INPUT	(0) Clears the alpha register like CLA does in the *HP-42S*.
	h P.FCN CLα	¬INPUT	
CLΣ	g CLΣ	DECM	(0) Clears the summation registers and releases the memory allocated for them.
CNST	h P.FCN CNST *n*	¬INPUT	(-1) Allows indirect addressing of the content at position n in CONST (see p. *134*).
^{C}CNST	CPX h X.FCN ^{C}CNST *n*	¬INPUT	(-2) Works like CNST but does a complex recall as described on p. *128*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
CNVG?	h TEST CNVG? s	DECM	(0) Checks for convergence by comparing x and y as determined by the lowest five bits of s. a) The very lowest two bits set the tolerance limit: 0 = 10^{-14}, 1 = 10^{-24}, 2 = 10^{-32}, 3 = Choose the best for the mode set, resulting in taking 0 for *SP* and 2 for *DP* (see Appendix H). b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the real numbers x and y relatively, 1 = compare them absolutely, 2 = check the absolute difference between the complex values $x + i\,y$ and $z + i\,t$, 3 = works as 0 so far. c) The top bit tells how special numbers are handled: 0 = NaN and infinities are considered converged, 1 = they are not considered converged. Now, $s = a + 4b + 16c$.
		Integer	(0) Tests for $x = y$.
COMB	f Cy,x	¬INPUT	(2) Returns the number of possible sets of y items taken x at a time. No item occurs more than once in a set, and different orders of the same x items are not counted separately. See Appendix I for the formula. Compare PERM.
ᶜCONJ	CPX h X.FCN ᶜCONJ	DECM	(1) Flips the sign of y, thus returning the complex conjugate of x_c.
	h CONST	¬INPUT	Catalog. See p. 125.
	h CONV	DECM	Catalog. See p. 125.
CORR	g r	DECM	(-1) Returns the correlation coefficient for the current statistical data and curve fitting model. See Appendix I for more.
COS	f COS	DECM	(1) Returns the cosine of the angle in X.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
COSH	f HYP COS	DECM	(1) Returns the hyperbolic cosine of x.
COV	h STAT COV	DECM	(-1) Returns the population covariance for two data sets. It depends on the fit model selected. See s_{XY} for the sample covariance and *Appendix I* for more.
^{C}CROSS	CPX h X.FCN 'CROSS	DECM	(2) Interprets x and y as Cartesian components of a first vector, and z and t as those of a second one, and returns $[x \cdot t - y \cdot z, 0, \ldots]$, dropping two stack levels.
DATE	h X.FCN DATE	DECM	(-1) Recalls the date from the real time clock into the numeric section in the format selected. See D.MY, M.DY, and Y.MD. In addition, DATE shows the day of week in the dot matrix. The function DATE of the *HP-12C* corresponds to DAYS+ in your *WP 34S* (see below).
DATE→	h X.FCN DATE→	DECM	(-2) Assumes x containing a date in the format selected and pushes its three components on the stack.
DAY	h X.FCN DAY	DECM	(1) Assumes x containing a date in the format selected and extracts the day.
DAYS+	h X.FCN DAYS+	DECM	(2) Adds x days on a date in $\mathbf{Y}$ in the format selected and displays the resulting date including the day of week in the same format as WDAY does. Works like DATE in the *HP-12C.*
DBLOFF	h MODE DBLOFF	All	(0) Toggles *DP* mode. Setting becomes effective in DECM only and is indicated by D in the dot matrix. See *Appendix H*.
DBLON	h MODE DBLON		
DBL?	h TEST DBL?	¬INPUT	(0) Checks if *DP* mode is turned on.
DBLR	h X.FCN DBLR	Integer	Double word length commands for remainder, multiplication and division. DBLR and DBL/ accept a double size dividend in $\mathbf{Y}$ and $\mathbf{Z}$ (most significant bits in $\mathbf{Y}$), the divisor in $\mathbf{X}$ as usual, and return the result in $\mathbf{X}$. DBL× takes x and y as factors as usual but returns their product in $\mathbf{X}$ and $\mathbf{Y}$ (most significant bits in $\mathbf{X}$). See the *HP-16C Owner's Handbook*, Section 4 (pp. 52ff).
DBL×	h X.FCN DBL×		
DBL /	h X.FCN DBL/		
DEC	h P.FCN DEC s	¬INPUT	(0) Decrements s by 1. Does not load $\mathbf{L}$ even for target address $\mathbf{X}$.
DECM	f H.d	¬INPUT	(0) Sets default decimal floating point mode.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
DECOMP	h X.FCN DECOMP	DECM	(-1) Decomposes x (after converting it into an improper fraction, if applicable), returning y/x = in top row and a stack [***denominator(x), numerator(x),*** …] . Reversible by division. **Example:** If **X** contains 2.25 then DECOMP will return $x = 4$ and $y = 9$, pushing previous content of **Y** to **Z** etc.
DEG	g DEG	DECM	(0) Sets angular mode to *degrees*.
DEG→	h X.FCN DEG→	DECM	(1) Takes x as *degrees* and converts them to the angular mode currently set.
DENANY	h MODE DENANY	All	(0) Sets default fraction format like in the *HP-35S*, allowing maximum precision in fraction display with the startup default – any denominator up to the value set by DENMAX may appear. **Example:** If DENMAX = 5 then DENANY allows denominators 1, 2, 3, 4, and 5.
DENFAC	h MODE DENFAC	All	(0) Sets 'factors of the maximum denominator', i.e. all integer factors of DENMAX may appear. **Example:** If DENMAX = 60 then DENFAC will allow denominators 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60. Now you know why 60 was a holy number in ancient Babylon.
DENFIX	h MODE DENFIX	All	(0) Sets fixed denominator format, i.e. the one and only denominator allowed is the value set by DENMAX.
DENMAX	h MODE DENMAX	All	(0) Works like /c in the *HP-35S*, but the maximum denominator settable is 9,999. It will be set to this value if $x < 1$ or $x > 9{,}999$ at DENMAX execution time. For $x = 1$ the current setting is recalled.
DET	h MATRIX DET	DECM	(1) Takes a *descriptor* of a square matrix in **X** and returns the determinant of the matrix. The matrix itself is not modified.
DISP	h MODE DISP *n*	All	(0) Changes the number of decimals shown while keeping the basic display format (FIX, SCI, ENG) as is. With ALL set, DISP will change the switchover point (see ALL).
^{C}DOT	CPX h X.FCN 'DOT	DECM	(2) Interprets x and y as Cartesian components of a first vector, and z and t as those of a second one, and returns $[x \cdot z + y \cdot t$, 0, …] , dropping two stack levels.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
dRCL	h P.FCN dRCL s	INPUT	(-1) Assumes the source contains *DP* data and recalls them as such. See *Appendix H*.
DROP	h X.FCN DROP	INPUT	Drops x. See p. 20 for details.
^{C}DROP	CPX h X.FCN 'DROP	DECM	Drops x_c. See p. 31 for details.
DSE	f DSE s	INPUT	(0) Given `cccccc.fffii` in the source, DSE decrements s by `ii`, skipping next program line if then `cccccc` ≤ `fff`. If s features no fractional part then `fff` is 0 and `ii` is set to 1. DSE does not load **L** even for target address **X**. Note that neither `fff` nor `ii` can be negative, and DSE makes only sense with `cccccc` > 0.
DSL	h P.FCN DSL s	INPUT	(0) Works like DSE but skips if `cccccc` < `fff`.
DSZ	h P.FCN DSZ s	INPUT	(0) Decrements s by 1, and skips the next step if $\lvert s \rvert < 1$ thereafter. Does not load **L** even for target address **X**. Known from the *HP-16C*.
D.MY	h MODE D.MY	All	(0) Sets the format for date display.
D→J	h X.FCN D→J	DECM	(1) Takes x as a date in the format set and converts it to a Julian day number according to JG…
D→R		DECM	(1) See the *catalog of conversions* for conversions from *degrees* to *radians*.
E3OFF	h MODE E3OFF	All	(0) Toggle the thousands separators for DECM (points or commas depending on radix setting).
E3ON	h MODE E3ON		
END	h P.FCN END	INPUT	(0) Last command in a routine and terminal for searching local labels as described on p. 67. Works like RTN in all other aspects.
ENG	h ENG n	INPUT	(0) Sets engineer's display format (see p. 40).
ENGOVR	h ENG ENTER↑	INPUT	(0) Defines that numbers exceeding the range displayable in ALL or FIX will be shown in engineer's format. Compare SCIOVR.
ENTER↑	ENTER↑	INPUT	(-1) Pushes x on the stack, disabling stack lift as usual. See pp. 20 and 31 for details.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
ENTRY?	h TEST ENTRY?	¬INPUT	(0) Checks the entry flag. This internal flag is set if: • any character is entered in alpha mode, or • any command is accepted for entry (be it via ENTER↑, a function key, or R/S with a partial command line).
erf	h X.FCN erf	DECM	(1) Returns the error function or its complement. See *Appendix I* for more.
erfc	h X.FCN erfc		
ERR	h P.FCN ERR *n*	¬INPUT	(0) Raises the error specified. The consequences are the same as if the respective error really occurred, so e.g. a running program will be stopped. See *Appendix C* for the respective error codes. Compare MSG.
EVEN?	h TEST EVEN?	¬INPUT	(0) Checks if x is integer and even.
e^x	f e^x	¬INPUT	(1)
	EXIT	All	See p. *120*.
ExpF	h MODE ExpF	All	(0) Selects the exponential curve fit model $R(x) = a_0 e^{a_1 x}$.
Expon	h PROB Expon	DECM	(1) Exponential distribution with the rate λ in **J**. See *Appendix I* for more. Expon^{-1} returns the survival time t_s for a given probability p in **X**, with λ in **J**.
Expon$_P$	h PROB Expon$_P$		
Expon$_u$	h PROB Expon$_u$		
Expon^{-1}	h PROB Expon^{-1}		
EXPT	h X.FCN EXPT	DECM	(1) Returns the exponent h of the number displayed $x = m \cdot 10^h$. Compare MANT.
e^x -1	h X.FCN e^x−1	DECM	(1) For $x \approx 0$, returns a more accurate result for the fractional part than e^x does.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
FAST	h MODE FAST	All	(0) Sets the processor speed to 'fast'. This is start-up default and is kept for fresh batteries. Compare SLOW.
FB	h X.FCN FB *n*	Integer	(1) Inverts ('flips') the specified bit in ***x*** .
FC?	h TEST FC? *n*	¬INPUT	(0) Tests if the flag specified is clear.
FC?C FC?F FC?S	h TEST FC?C *n* etc.	¬INPUT	(0) Tests if the flag specified is clear. Clears, flips, or sets this flag after testing, respectively.
FF	h P.FCN FF *n*	¬INPUT	(0) Flips the flag specified.
FIB	h X.FCN FIB	Integer DECM	(1) Returns the Fibonacci number. (1) Returns the extended Fibonacci number. See *App. I* for more.
FILL	g FILL	¬INPUT	(0) Copies ***x*** to all stack levels. See p. *31* for ^{C}FILL.
FIX	h FIX *n*	¬INPUT	(0) Sets fixed point display format (see p. *40*).
FLASH?	h P.FCN FLASH?	¬INPUT	(-1) Returns the number of free *words* in *FM*.
FLOOR	h X.FCN FLOOR	¬INPUT	(1) Returns the largest integer ≤ ***x*** .
FP	g FP	¬INPUT	(1) Returns the fractional part of ***x*** .
FP?	h TEST FP?	¬INPUT	(0) Tests ***x*** for having a nonzero fractional part.
FRACT	h MODE FRACT	All	(0) Sets fraction mode like in the *HP-35S*, but keeps the format as set by PROFRC or IMPFRC.
FS?	h TEST FS? *n*	¬INPUT	(0) Tests if the flag specified is set.
FS?C FS?F FS?S	h TEST FS?C *n* etc.	¬INPUT	(0) Tests if the flag specified is set. Clears, flips, or sets this flag after testing, respectively.
$F_P(x)$	h PROB Fp(x)	DECM	(1) *Fisher's* F-distribution. F(x) equals 1 – Q(F) , $F_u(x)$ equals Q(F) and $F^{-1}(p)$ equals F_P in the *HP-21S*. The degrees of freedom are specified in **J** and **K**. See *Appendix I* for more.
$F_u(x)$	h PROB Fu(x)		
F(x)	h PROB F(x)		
$F^{-1}(p)$	h PROB F-1(p)		

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
f'(x)	h P.FCN f'(x) ***label***	DECM	Returns the first derivative of the function ***f(x)*** at position ***x***. This ***f(x)*** must be specified in a routine starting with LBL ***label***. On return, **Y**, **Z**, and **T** will be cleared and the position ***x*** will be in **L**. **ATTENTION:** f'(x) will look for a user routine labeled 'δx', returning a fixed step size ***dx*** in **X**. If that routine is not defined, ***dx*** = *0.1* is set for default (arbitrary, but it had to be specified). Then, f'(x) will evaluate ***f(x)*** at ten points equally spaced in the interval $x \pm 5\,dx$. If you expect any irregularities within this interval, change ***dx*** to exclude them.
f"(x)	h P.FCN f"(x) ***label***	DECM	Works like f'(x) but returns the second derivative. The remarks above apply in analogy.
GCD	h X.FCN GCD	INPUT	(2) Returns the Greatest Common Divisor of ***x*** and ***y*** [52]. This will always be positive.
gCLR	h P.FCN gCLR ***s***	INPUT	(0) Clears the pixel at position ***x***, ***y*** in the graphic block starting at register address ***s***. Valid ranges are $0 \le x \le w-1$ and $0 \le y \le h-1$. Pixel 0, 0 is top left. See gDIM for more.
g_d	h X.FCN g_d	DECM	(1) Returns the Gudermann function or its inverse. See *Appendix I* for details.
g_d^{-1}	h X.FCN g_d^-1		
gDIM	h P.FCN gDIM ***s***	INPUT	(0) Initiates a set of registers (called a *graphic block*) starting at address ***s***, allowing for graphic data featuring ***x*** (≤ 166) pixel columns and ***y*** pixel rows. For $x \le 0$, the width ***w*** will be set to 166. For $y \le 0$, the height ***h*** will be set to 8. The first two *bytes* are reserved to take ***w*** and $\check{h} = floor\left(\frac{h+7}{8}\right)$. The number of registers needed for the set is $floor\left(\frac{w \cdot \check{h}+9}{8}\right)$ in startup standard mode. E.g. 21 registers are required for maximum width and standard height. The command can be exactly emulated in integer mode by storing $256 \cdot \check{h} + w$ in the first register and clearing the rest. See FLOOR and ⎙PLOT.
gDIM?	h P.FCN gDIM? ***s***	INPUT	(-2) Assumes a graphic block starting at address ***s*** and recalls ***h*** and ***w*** for it. See gDIM for more.

[52] GCD translates to "ggT" in German.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Geom	h PROB Geom	DECM	(1) Geometric distribution: The cdf returns the probability for a first success after $m = x$ Bernoulli experiments. The probability p_0 for a success in each such experiment must be specified in **J**. See *Appendix I* for more. Geom $^{-1}$ returns the number of failures f before 1st success for given probabilities p in **X** and p_0 in **J**.
Geom$_p$	h PROB Geomp		
Geom$_u$	h PROB Geomu		
Geom $^{-1}$	h PROB Geom-1		
gFLP	h P.FCN gFLP *s*	INPUT	(0) Flips or tests the pixel at position x, y in the graphic block starting at address s. See gCLR for more.
gPIX?	h TEST gPIX? *s*		
gPLOT	h P.FCN gPLOT *s*	INPUT	(0) Displays the top left sector of the graphic block (starting at address s) in the dot matrix section of the *LCD*. See gDIM for more.
GRAD	g GRAD	DECM	(0) Sets angular mode to *gon* (*grad*).
GRAD→	h X.FCN GRAD→	DECM	(1) Takes x as given in *gon* (*grad*) and converts them to the angular mode currently set.
gSET	h P.FCN gSET *s*	INPUT	(0) Sets the pixel at position x, y in the graphic block starting at address s. See gCLR for more.
GTO	h GTO ***label***	STO	(0) Inserts an unconditional branch to ***label*** .
		STO, INPUT	(0) Positions the program pointer to ***label*** .
	h GTO . A , B , C , or D	INPUT	(0) Positions the program pointer … (not programmable) … to one of these labels, if defined.
	h GTO . ***nnn***		… to step ***nnn*** .
	h GTO . ▲		… directly after previous END, going to the top of current program.
	h GTO . ▼		… directly after next END, going to the top of next program.
	h GTO . .		… to step 000, i.e. top of *RAM*.
GTOα	h P.FCN GTOα	INPUT	(0) Takes the first three characters of ***alpha*** (or all if there are fewer than three) as a label and positions the program pointer to it.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
H_n	h X.FCN Hn	DECM	(2) Hermite polynomials for probability (H_n) and for physics (H_{np}). See *Appendix I* for details.
H_{np}	h X.FCN Hnp		
H.MS	f H.MS	DECM	(1) Assumes **X** containing decimal *hours* or *degrees*, and displays them converted in the format `hhhh°mm'ss.dd"` temporarily as shown on p. *44*.
H.MS+	h X.FCN H.MS+	DECM	(2) Assumes **X** and **Y** containing times or *degrees* in the format `hhhh.mmssdd`, and adds or subtracts them, respectively.
H.MS–	h X.FCN H.MS–		
IDIV	h X.FCN IDIV *s*	INPUT	(2) Integer division, working like / IP in DECM and like / in integer modes.
IMPFRC	g d/c	INPUT	(1) Sets fraction mode allowing improper fractions in display (e.g. $^5/_3$ instead of $1\,^2/_3$). Converts x according to the settings by DEN... into an improper fraction if possible. Absolute decimal equivalents of x must not exceed 100,000. Compare PROFRC.
INC	h P.FCN INC *s*	INPUT	(0) Increments s by 1. Does not load **L** even for target address **X**.
INTM?	h TEST INTM?	INPUT	(0) Tests if your *WP 34S* is in an integer mode.
INT?	h TEST INT?	INPUT	(0) Tests x for being an integer, i.e. having a fractional part equal to zero. Compare FP?.
IP	f IP	INPUT	(1) Returns the integer part of x .
iRCL	h X.FCN iRCL *s*	INPUT	(-1) Assumes the source contains integer data and recalls them as such. See p. *159*.
ISE	h P.FCN ISE *s*	INPUT	(0) Works like ISG but skips if `cccccc` ≥ `fff` .
ISG	g ISG *s*	INPUT	(0) Given `cccccc.fffii` in the source , ISG increments s by `ii`, skipping next program line if then `cccccc` > `fff` . If s has no fractional part then `fff` = 0 , of course, and `ii` is set to 1. ISG does not load **L** even for target address **X**. Note that neither `fff` nor `ii` can be negative, but `cccccc` can.
ISZ	h P.FCN ISZ *s*	INPUT	(0) Increments s by 1, skipping next program line if then $\lvert s\rvert < 1$. Does not load **L** even for target address **X**. Known from the *HP-16C*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Iβ	h X.FCN Iβ	DECM	Returns the regularized (incomplete) beta or gamma function (one of two kinds). See *Appendix I* for details.
$I\Gamma_p$	h X.FCN IΓp		
$I\Gamma_q$	h X.FCN IΓq		
JG1582	h MODE JG1582	All	(0) These two commands reflect different dates the Gregorian calendar was introduced in different large areas of the world. D→J and J→D will be calculated accordingly. See p. 39.
JG1752	h MODE JG1752		
J→D	h X.FCN J→D	DECM	(1) Takes x as a Julian day number and converts it to a date according to JG... in the format selected
KEY?	h TEST KEY? *s*	INPUT	(0) Tests if a key was pressed while a program was running or paused. If no key was pressed in that interval, the next program step after KEY? will be executed, else it will be skipped and the code of said key will be stored in the address specified. Key codes reflect the rows and columns on the keyboard (see p. 71).
KTP?	h P.FCN KTP? *s*	INPUT	(-1) Assumes a key code in the address specified (see KEY?). Checks this code and returns the key type: • 0 … 9 if it corresponds to a digit 0 … 9 , • 10 if it corresponds to . , EEX , or +/− , • 11 if it corresponds to f , g , or h , • 12 if it corresponds to any other key. May help in user interaction with programs.
LASTx	RCL L	INPUT	(-1) See pp. 20 and 31 for details. In fact, this command will be recorded as RCL L in programs.
LBL	f LBL *label*	STO	(0) Identifies programs and routines for execution and branching. Read more about specifying labels on p. 67f.
LBL?	h TEST LBL? *label*	INPUT	(0) Tests for the existence of the label specified, anywhere in program memory. See LBL for more.
LCM	h X.FCN LCM	INPUT	(2) Returns the Least Common Multiple of x and y [53]. This will always be positive.
LEAP?	h TEST LEAP?	DECM	(0) Takes x as a date in the format selected, extracts the year, and tests for a leap year.

[53] LCM translates to "kgV" in German.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
LgNrm	h PROB LgNrm	DECM	(1) Lognormal distribution with $\mu = \ln \bar{x}_g$ specified in **J** and $\sigma = \ln \varepsilon$ in **K**. See x̄g and ε below. See *Appendix I* for details. LgNrm^{-1} returns ***x*** for a given probability ***p*** in **X**, with ***μ*** in **J** and ***σ*** in **K**.
LgNrm$_P$	h PROB LgNrmP		
LgNrm$_u$	h PROB LgNrmu		
LgNrm^{-1}	h PROB LgNrm-1		
LINEQS	h X.FCN LINEQS	INPUT	(3) Solves a system of linear equations $(Z)\cdot\vec{x} = \vec{y}$. Takes a base register number in **X**, a vector *descriptor* in **Y**, and a descriptor of a square matrix in **Z**. Returns the filled in vector descriptor in **X**.
LinF	h MODE LinF	All	(0) Selects the linear curve fit model $R(x) = a_0 + a_1 x$.
LJ	h X.FCN LJ	Integer	(-1) Left justifies a bit pattern within its word size as in the *HP-16C*: The stack will lift, placing the left-justified word in **Y** and the count (number of bit-shifts necessary to left justify the word) in **X**. **Example** for word size 8: 10110_2 LJ results in $x = 3$ and $y = 10110000_2$.
LN	g LN	INPUT	(1) Returns the natural logarithm of ***x***.
L_n	h X.FCN Ln	DECM	(2) Laguerre polynomials. See *Appendix I* for more.
LN1+x	h X.FCN LN1+x	DECM	(1) For $x \approx 0$, this returns a more accurate result for the fractional part than $\ln(x)$ does.
$L_n\alpha$	h X.FCN Lnα	DECM	(3) Laguerre's generalized polynomials. See *Appendix I* for more.
LN β	h X.FCN LNβ	DECM	(2) Returns the natural logarithm of Euler's Beta function. See *β*.
LN Γ	h X.FCN LNΓ	DECM	(1) Returns the natural logarithm of *Γ(x)* . Allows also for calculating really large factorials. **Example**: What is 5432*!* ? Remember *Γ*(***x*** + 1) = ***x!*** So, entering 5433 LNΓ 10 LN / returns 17 931.480374 being the decadic logarithm of the result; now calling FP 10^x returns its mantissa 3.0225535984. Thus 5432*!* = $3.02\cdot10^{17\,931}$.
LOAD	h P.FCN LOAD	INPUT	Restores the entire backup from *FM*, i.e. executes LOADP, LOADR, LOADSS, LOADΣ, and returns Restored. Not programmable. Compare SAVE. See the commands mentioned below and *Appendix A* for more.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
LOADP	h P.FCN LOADP	¬INPUT	(0) Loads the complete program memory from the backup and appends it to the programs already in *RAM*. This will only work if there is enough space – else an error will be thrown. Not programmable.
LOADR	h P.FCN LOADR	¬INPUT	(0) Recovers numbered general purpose registers from the backup (see SAVE). Lettered registers will not be recalled. The number of registers copied is the minimum of the registers held in the backup and *RAM* at execution time.
LOADSS	h P.FCN LOADSS	¬INPUT	(0) Recovers the system state from the backup. See *Appendix B* for more.
LOADΣ	h P.FCN LOADΣ	¬INPUT	(0) Recovers the summation registers from the backup. Throws an error if there are none. See *Appendix B* for more.
LocR	h P.FCN LocR ***n***	¬INPUT	(0) Allocates ***n*** local registers (≤ 144) and 16 local flags for the current program. See p. *69*.
LocR?	h P.FCN LocR?	¬INPUT	(-1) Returns the number of local registers currently allocated.
LOG₁₀	g LG	¬INPUT	(1) Returns the logarithm of ***x*** for base 10.
LOG₂	g LB	¬INPUT	(1) Returns the logarithm of ***x*** for base 2.
LogF	h MODE LogF	All	(0) Selects the logarithmic curve fit model $R(x) = a_0 + a_1 \ln x$.
Logis	h PROB Logis	DECM	(1) Logistic distribution with ***μ*** given in **J** and ***s*** in **K**. See *Appendix I* for details.
Logis_P	h PROB Logis▸		
Logis_u	h PROB Logis▾		
Logis^{-1}	h PROB Logis⁻¹		
LOG_x	g LOG_x	¬INPUT	(2) Returns the logarithm of ***y*** for the base ***x*** .
$^c\text{LOG}_x$	CPX g LOG_x	DECM	(2) Returns the complex logarithm of $z + i\,t$ for the complex base $x + i\,y$.
LZOFF	h MODE LZOFF	All	(0) Toggles leading zeros like flag 3 does in the *HP-16C*. Relevant in bases 2, 4, 8, and 16 only.
LZON	h MODE LZON		

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
L.R.	h STAT L.R.	DECM	(-2) Returns the parameters a_1 and a_0 of the fit curve through the data points accumulated in the summation registers, according to the curve fit model selected (see LINF, EXPF, POWERF, and LOGF). For a straight line (LINF), a_0 is the y-intercept and a_1 the slope. See Appendix I for more.
MANT	h X.FCN MANT	DECM	(1) Returns the mantissa m of the number displayed $x = m \cdot 10^h$. Compare EXPT.
MASKL	h X.FCN MASKL *n*	Integer	(-1) Work like MASKL and MASKR on the *HP-16C*, but with the mask length (or its address) following the command instead of taken from **X**. Thus, the mask is pushed on the stack. **Example**: For WSIZE 8, MASKL 3 returns a mask word 11100000_2. Use it e.g. for extracting the three most significant bits from an arbitrary *byte* by AND.
MASKR	h X.FCN MASKR *n*		
	h MATRIX	DECM	Catalog. See p. 125.
MAX	h X.FCN MAX	¬INPUT	(2) Returns the maximum of x and y .
MEM?	h P.FCN MEM?	¬INPUT	(-1) Returns the number of free *words* in program memory, taking into account the local registers allocated.
MIN	h X.FCN MIN	¬INPUT	(2) Returns the minimum of x and y .
MIRROR	h X.FCN MIRROR	Integer	(1) Reflects the bit pattern in x (e.g. 00010111_2 would become 11101000_2 for word size 8).
MOD	h X.FCN MOD	¬INPUT	(2) Returns $y \bmod x$ (see p. 56 for examples). Compare RMDR.
	h MODE	All	Catalog. See p. 125.
MONTH	h X.FCN MONTH	DECM	(1) Assumes x containing a date in the format selected and extracts the month.
MROW+×	h MATRIX MROW+×	DECM	(0) Takes a matrix *descriptor* x, a destination row number y, a source row number z, and a real number t. It multiples each element m_{zi} of (X) by t and adds it to m_{yi} . The stack remains unchanged. M.ROW+× is similar to *PPC M3*.
MROW×	h MATRIX MROW×	DECM	(0) Takes a matrix *descriptor* x, a row number y, and a real number z. It multiples each element m_{yi} of (X) by z. The stack remains unchanged. M.ROW× is similar to *PPC M2*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
MROW⇆	h MATRIX MROW⇆	DECM	(0) Takes a matrix *descriptor* x and two row numbers y and z. It swaps the contents of rows y and z in (X). The stack remains unchanged. M.ROW⇆ is similar to *PPC M1*.
MSG	h P.FCN MSG *n*	¬INPUT	(0) Throws the error message specified. This will be a *temporary message*. See *Appendix C* for the respective error codes. Compare ERR.
M+×	h MATRIX M+×	DECM	(3) Takes two matrix *descriptors* x and y, and a real number z. Returns $(X) + (Y) \cdot z = (X)$. Thus a scalar multiple of one matrix is added to another matrix. The multiply adds are done in internal high precision and results should be exactly rounded.
M^{-1}	h MATRIX M-¹	DECM	(0) Takes a *descriptor* of a square matrix in **X** and inverts the matrix in-situ. Doesn't alter the stack.
M-ALL	h MATRIX M–ALL	DECM	(1) Takes a matrix *descriptor* x [54] and returns a value suitable for ISG or DSL looping in that matrix. The loop shall process all elements in (X). The loop counter is for DSL if the descriptor was negative and for ISG otherwise.
M-COL	h MATRIX M–COL	DECM	(2) Takes a matrix *descriptor* x (positive or negative as for M-ALL) and a column number y. Returns a loop counter for processing all elements m_{iy} of that matrix column only.
M-DIAG	h MATRIX M–DIAG	DECM	(1) Takes a matrix *descriptor* x (as for M-ALL) and returns a loop counter. The loop shall processes all elements along the matrix diagonal, i.e. all elements m_{ii} in (X).
M-ROW	h MATRIX M–ROW	DECM	(2) Takes a matrix *descriptor* x (as for M-ALL) and a row number y. Returns a loop counter for processing all elements m_{yi} of that matrix row only.
M×	h MATRIX M×	DECM	(3) Takes two matrix *descriptors* y and z, and the integer part of x as the base address of the result. Returns $(Z) \cdot (Y) = (X)$. All calculations are done in internal high precision (39 digits). The fractional part of x is updated to match the resulting matrix – no overlap checking is performed.
M.COPY	h MATRIX M.COPY	DECM	(2) Takes a matrix *descriptor* y and a base register number x. Copies the matrix (Y) into registers starting at Rx. Returns a properly formatted matrix descriptor in **X**.

[54] That descriptor will be saved in **L** as usual. This applies for the other matrix operations as well.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
M.DY	h MODE M.DY	All	(0) Sets the format for date display.
M.IJ	h MATRIX M.IJ	DECM	Takes a matrix *descriptor* x and a register number y. Returns the column that register represents in **Y** and the row in **X**. M.IJ is similar to *PPC M4*. Compare M.REG.
M.LU	h MATRIX M.LU	DECM	(1) Takes a *descriptor* of a square matrix in **X**. Transforms (X) into its LU decomposition in-situ. The value in **X** is replaced by a descriptor that defines the pivots that were required to calculate the decomposition. The most significant digit is the pivot for the first diagonal entry, the next most significant for the second and so forth.
M.REG	h MATRIX M.REG	DECM	(3) Takes a matrix *descriptor* x, a row number y, and a column number z. M.REG returns the register number in **X**. It is similar to *PPC M5*. Compare M.IJ.
M.SQR?	h TEST M.SQR?	DECM	(0) Tests a matrix *descriptor* x and returns true if the matrix is square.
NAND	h X.FCN NAND	INPUT	(2) Works in analogy to AND. See *there*.
NaN?	h TEST NaN?	INPUT	(0) Tests x for being 'Not a Number'.
nBITS	h X.FCN nBITS	Integer	(1) Counts bits set in x like #B does on the *HP-16C*.
nCOL	h MATRIX nCOL	DECM	(1) Takes a matrix *descriptor* x and returns the number of columns of (X).
NEIGHB	h X.FCN NEIGHB	DECM	(2) Returns the nearest machine-representable number to x in the direction toward y in the mode set [55]. For $x < y$ (or $x > y$), this is the machine successor (or predecessor) of x; for $x = y$ it is y.
		Integer	(2) Returns $x + 1$ for $x < y$, y for $x = y$, or $x - 1$ for $x > y$.
NEXTP	h X.FCN NEXTP	INPUT	(1) Returns the next prime number greater than x.
NOP	h P.FCN NOP	INPUT	(0) 'Empty' step FWIW.
NOR	h X.FCN NOR	INPUT	(2) Works in analogy to AND. See *there*.

[55] You may find NEIGHB useful investigating numeric stability. See NEIGHBOR in the HP-71 Math Pac.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Norml	h PROB Norml	DECM	(1) Normal distribution with an arbitrary mean μ given in **J** and a standard deviation σ in **K**. See *Appendix I* for details. Norml^{-1} returns x for a given probability p in **X**, with μ in **J** and σ in **K**.
Norml$_P$	h PROB Norml$_P$		
Norml$_u$	h PROB Norml$_u$		
Norml^{-1}	h PROB Norml^{-1}		
NOT	h NOT	Integer	(1) Inverts x bit-wise as on the *HP-16C*.
		DECM	(1) Returns 1 for $x = 0$, and 0 for $x \neq 0$.
nROW	h MATRIX nROW	DECM	(1) Takes a matrix *descriptor* x and returns the number of rows of (X).
nΣ	h SUMS nΣ	DECM	(-1) Recalls the number of accumulated data points.
ODD?	h TEST ODD?	INPUT	(0) Checks if x is integer and odd.
OFF	h OFF	STO	(0) Inserts a step to turn your *WP 34S* off under program control.
		STO	(0) Turns your *WP 34S* off.
OR	h OR	INPUT	(2) Works in analogy to AND. See *there*.
PERM	g Py,x	INPUT	(2) Returns the number of possible arrangements of y items taken x at a time. No item occurs more than once in an arrangement, and different orders of the same x items are counted separately. See *Appendix I* for the formula. Compare COMB.
P_n	h X.FCN P_n	DECM	(1) Legendre polynomials. See *App. I* for more.
Poiss	h PROB Poiss	DECM	(1) Poisson distribution with the number of successes g in **X**, the gross error probability p_0 in **J**, and the sample size n in **K**. The Poisson parameter $\lambda = n \cdot p_0$ is calculated automatically. See Poisλ below. Poiss^{-1} returns the maximum number of successes m for a given probability p in **X**, p_0 in **J** and n in **K**.
Poiss$_P$	h PROB Poiss$_P$		
Poiss$_u$	h PROB Poiss$_u$		
Poiss^{-1}	h PROB Poiss^{-1}		
Poisλ	h PROB Poisλ	DECM	(1) Poisλ works like Poiss but with λ in **J** and without using **K**. See *Appendix I* for more. Poisλ^{-1} returns m for a given probability p in **X** and λ in **J**.
Poisλ$_P$	h PROB Poisλ$_P$		
Poisλ$_u$	h PROB Poisλ$_u$		
Poisλ^{-1}	h PROB Poisλ^{-1}		

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
PopLR	h P.FCN PopLR	¬INPUT	(0) Pops the local registers allocated to the current routine without returning. See LOCR and RTN.
PowerF	h MODE PowerF	All	(0) Selects the power curve fit model $R(x) = a_0 x^{a_1}$.
PRCL	h P.FCN PRCL	¬INPUT	(0) Copies the current program (from *FM* or *RAM*) and appends it to *RAM*, where it can then be edited (see p. 72). Allows duplicating programs in *RAM*. Will only work with enough space at destination. By recalling a library routine from *FM*, editing it, and PSTO you may modify this part of the *FM* library.
	RCL	CAT open	
PRIME?	h TEST PRIME?	¬INPUT	(0) Checks if the absolute value of the integer part of ***x*** is a prime. The method is believed to work for integers up to $9 \cdot 10^{18}$.
	h PROB	DECM	Catalog. See p. 125.
PROFRC	f a b/c	¬INPUT	(1) Sets fraction mode like in the *HP-35S*, allowing only proper fractions or mixed numbers in display. Converts ***x*** according to the settings by DEN… into a proper fraction, e.g. 1.25 or $^5/_4$ into $1\,^1/_4$. Absolute decimal equivalents of ***x*** must not exceed 100,000. Compare IMPFRC.
PROMPT	h P.FCN PROMPT	¬INPUT	(0) Displays ***alpha*** and stops program execution. See p. 69 for more.
PSE	h PSE ***n***	¬INPUT	(0) With a program running, refreshes the display and pauses program execution for ***n*** ticks (see TICKS), with $0 \le n \le 99$. The pause will terminate early when you press a key.
PSTO	h P.FCN PSTO	¬INPUT	(0) Copies the current program from *RAM* and appends it to the *FM* library. Not programmable. The program must have at least one LBL statement with an alphanumeric global label (preferably at its beginning). If a program with the same label already exists in the library it will be deleted first. Alphanumeric labels contained in *FM* may be browsed by CAT (see p. 126) and called by XEQ. By recalling a library routine from *FM* via PRCL, editing it, and PSTO you may modify this part of the *FM* library.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
PUTK	h P.FCN PUTK *s*	INPUT	(0) Assumes a key code in the address specified. Stops program execution, takes said code and puts it in the keyboard buffer resulting in immediate execution of the corresponding call. R/S is required to resume program execution. May help in user interaction with programs.
	h P.FCN	INPUT	Catalog. See p. 125.
RAD	g RAD	DECM	(0) Sets current angular mode to *radians*.
RAD→	h X.FCN RAD→	DECM	(1) Takes ***x*** as *radians* and converts it to the angular mode currently set.
RAN#	f RAN#	DECM	(-1) Returns a random number between 0 and 1 like RAN does in the *HP-42S*.
		Integer	(-1) Returns a random bit pattern for the word size set.
RCL	RCL *s*	INPUT	(-1) See the *addressing table above* for ^C^RCL.
RCLM	RCL MODE *s*	INPUT	(0) Recalls modes stored by STOM (see p. 37). No need pressing h here.
RCLS	h P.FCN RCLS *s*	INPUT	Recalls 4 or 8 values from a set of registers starting at address ***s***, and pushes them on the stack. This is the converse command of STOS.
RCL+	RCL + *s*	INPUT	(1) Recalls ***s***, executes the specified operation and pushes the result on the stack. E.g. RCL–12 subtracts ***r12*** from ***x*** and displays the result (acting like RCL 1 2 –, but without losing a stack level). In analogy, ^C^RCL–12 subtracts ***r12*** from ***x*** and ***r13*** from ***y***.
RCL–	RCL – *s*		
RCL×	RCL × *s*		
RCL/	RCL / *s*		
RCL↑	RCL ▲ *s*	INPUT	(1) RCL↑ (↓) replaces ***x*** with the maximum (minimum) of ***s*** and ***x***.
RCL↓	RCL ▼ *s*		
RDP	h X.FCN RDP *n*	DECM	(1) Rounds ***x*** to ***n*** decimal places ($0 \le n \le 99$), taking the RM setting into account. See RM and compare RSD.
RDX,	h MODE RDX,	All	(0) Sets the decimal mark to a comma.
	h ./,	DECM	(0) Toggles the radix mark.
RDX.	h MODE RDX.	All	(0) Sets the decimal mark to a point.
REALM?	h TEST REALM?	INPUT	(0) Tests if your *WP 34S* is in real mode (DECM).

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
RECV	h P.FCN RECV	¬INPUT	(0) Prepares your *WP 34S* for receiving data via serial I/O. See SEND… and *Appendix A* for more.
REGS	h MODE REGS *n*	All	(0) Specifies the number of global general purpose registers wanted. With REGS 100 you get the default state (**R00** – **R99**), REGS 0 leaves not even a single such register for use.
REGS?	h P.FCN REGS?	¬INPUT	(-1) Returns the number of global general purpose registers allocated (0 … 100).
RESET	h P.FCN RESET	¬INPUT	If confirmed, executes CLALL and resets all modes to start-up default, i.e. 24h, 2COMPL, ALL 0, DBLOFF, DEG, DENANY, DENMAX 0, D.MY, E3ON, LinF, LocR 0, LZOFF, PROFRC, RDX., REGS 100, SCIOVR, SEPON, SSIZE4, WSIZE 64, and finally DECM. See these commands for more. Not programmable.
RJ	h X.FCN RJ	Integer	(-1) Right justifies, in analogy to LJ on the *HP-16C*. **Example**: 101100_2 RJ results in $y = 1011_2$ and $x = 2$. See LJ.
RL	h X.FCN RL *n*	Integer	(1) Works like n consecutive RLs / RLCs on the *HP-16C*, similar to RLn / RLCn there. For RL, $0 \le n \le 63$. For RLC, $0 \le n \le 64$. RL 0 / RLC 0 execute as NOP, but load **L**. See p. 56 for more.
RLC	h X.FCN RLC *n*		
RM	h MODE RM *n*	All	(0) Sets floating point rounding mode. This is only used when converting from the high precision internal format to packed real numbers. It will not alter the display nor change the behavior of ROUND. The following modes are supported: 0: round half even: ½ = 0.5 rounds to next even number (default). 1: round half up: 0.5 rounds up ('business-man's rounding' [56]). 2: round half down: 0.5 rounds down. 3: round up: rounds away from 0. 4: round down: rounds towards 0 (truncates). 5: ceiling: rounds towards +∞. 6: floor: rounds towards –∞.
RMDR	h RMDR	¬INPUT	(2) Returns the remainder of a division. Equals RMD on the *HP-16C* but works for real numbers as well. See p. 56 for examples. Compare MOD.

[56] Translates to "kaufmännische Rundung" in German.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
RM?	h P.FCN RM?	¬INPUT	(-1) Returns the floating point rounding mode set. See RM for more.
ROUND	g RND	¬INPUT	(1) Rounds x using the current … display format like RND does in the *HP-42S*.
		FRC	… denominator like RND does in the *HP-35S* fraction mode.
ROUNDI	h X.FCN ROUNDI	¬INPUT	(1) Rounds x to next integer. ½ rounds to 1.
RR	h X.FCN RR n	Integer	(1) Works like n consecutive RRs / RRCs on the *HP-16C*, similar to RRn / RRCn there. For RR, $0 \le n \le 63$. For RRC, $0 \le n \le 64$. RR 0 / RRC 0 execute as NOP, but load **L**. See p. 56 for more.
RRC	h X.FCN RRC n		
RSD	h X.FCN RSD n	DECM	(1) Rounds x to n significant digits, taking the RM setting into account. See RM and compare RDP.
RTN	g RTN	STO	(0) Last command in a typical routine. Pops the local data (like PopLR) and returns control to the calling routine in program execution, i.e. moves the program pointer one step behind the XEQ instruction that called said routine. If there is none, program execution halts and the program pointer is set to step 000.
		¬STO	(0) Resets the program pointer to the beginning of current program. If the current program is in *FM*, the program pointer will be set to step 000 in *RAM*.
RTN+1	h P.FCN RTN+1	¬INPUT	(0) Works like RTN, but moves the program pointer two steps behind the XEQ instruction that called said routine. Halts if there is none.
R-CLR	h P.FCN R−CLR	DECM	(0) Interprets x in the form `sss.nn`. Clears ***nn*** registers starting with address ***sss***. If ***nn*** = 0, it will clear the maximum available. **Example:** For $x = 34.567$, R-CLR will clear **R34** through **R89**. **ATTENTION for *nn* = 0 :** For $sss \in [0;99]$, clearing will stop at the highest allocated global numbered register. For $sss \in [100;111]$, clearing will stop at **K**. For $sss \ge 112$, clearing will stop at the highest allocated local register.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
R-COPY	h P.FCN R-COPY	DECM	(0) Interprets x in the form `sss.nnddd`. Takes ***nn*** registers starting with address ***sss*** and copies their contents to ***ddd*** etc. If ***nn*** = 00, it will take the maximum available. **Example:** For $x = 7.03045678$, ***r07, r08, r09*** will be copied into **R45, R46, R47**, respectively. For $x < 0$, R-COPY will take ***nn*** registers from *FM* instead, starting with register number |***sss***|. Destination will be in *RAM* always. **ATTENTION:** The advice at R-CLR applies, but with 'clearing' replaced by 'copying'.
R-SORT	h P.FCN R-SORT	DECM	(0) Interprets x in the form `sss.nn`. Sorts the contents of ***nn*** registers starting with address ***sss***. If ***nn*** = 0, it will sort the maximum available. **Example:** Assume $x = 49.0369$, ***r49*** = 1.2, ***r50*** = –3.4, and ***r51*** = 0; then R-SORT will return ***r49*** = -3.4, ***r50*** = 0, and ***r51*** = 1.2. **ATTENTION:** The advice at R-CLR applies, but with 'clearing' replaced by 'sorting'.
R-SWAP	h P.FCN R-SWAP	DECM	(0) Works like R-COPY but swaps the contents of source and destination registers.
R→D		DECM	(1) See the *catalog of conversions* for conversions of *radians* to *degrees*.
R↑	h R↑	INPUT	Rotates the stack contents one level up or down, respectively. See pp. *20* and *31* for details.
R↓	R↓		
s	g s	DECM	(-2) Takes the statistical sums accumulated, calculates the sample standard deviations s_y and s_x and pushes them on the stack. See *Appendix I* for the formula.
SAVE	h P.FCN SAVE	INPUT	(0) Saves user program space, registers and system state to *FM*, and returns Saved. Recall your backup by the different flavors of LOAD. Not programmable. See *Appendix A* for more.
SB	h X.FCN SB ***n***	Integer	(1) Sets the specified bit in x.
SCI	h SCI ***n***	INPUT	(0) Sets scientific display format (see p. *40*).
SCIOVR	h SCI ENTER↑	INPUT	(0) Defines that numbers exceeding the range displayable in ALL or FIX will be shown in scientific format. Compare ENGOVR, see RESET.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
SDL	h X.FCN SDL *n*	DECM	(1) Shifts digits left (right) by ***n*** decimal positions, equivalent to multiplying (dividing) ***x*** by 10^n . Compare SL and SR for integers.
SDR	h X.FCN SDR *n*		
SEED	h STAT SEED	DECM	(0) Stores a seed for random number generation.
SENDA	h P.FCN SENDA etc.	INPUT	(0) SENDA sends all *RAM* data, SENDP the program memory, SENDR the global general purpose registers, and SENDΣ the summation registers, respectively, to the device connected via serial I/O. See RECV and *Appendix A* for more.
SENDP			
SENDR			
SENDΣ			
SEPOFF	h MODE SEPOFF	All	(0) Toggle the digit group separators for integers. Points or commas will be displayed every … … four digits in bases 2 and 4, … two digits in base 16, … three digits in all other integer bases.
	h ./,	Integer	
SEPON	h MODE SEPON	All	
SERR	h STAT SERR	DECM	(-2) Takes the statistical sums accumulated, calculates and returns the standard errors (i.e. the standard deviations of $\bar{x}$ and $\bar{y}$).
$SERR_w$	h STAT $SERR_w$	DECM	(-1) Returns the standard error for weighted data, i.e. the standard deviation of $\bar{x}_w$.
SETCHN	h MODE SETCHN	All	(0) Sets some regional preferences (see p. *39*).
SETDAT	h MODE SETDAT	All	(0) Sets the date for the real time clock (the emulator takes this information from the PC clock).
SETEUR	h MODE SETEUR etc.	All	(0) Sets some regional preferences (see p. *39*).
SETIND			
SETJPN			
SETTIM	h MODE SETTIM	All	(0) Sets the time for the real time clock (the emulator takes this information from the PC clock).
SETUK	h MODE SETUK etc.	All	(0) Sets some regional preferences (see p. *39*).
SETUSA			
SF	f SF *n*	INPUT	(0) Sets the flag specified.
SHOW	g SHOW	INPUT	Browser. See p. *125*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
SIGN	h X.FCN SIGN	¬INPUT	(1) Returns 1 for $x > 0$, –1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to $\mathrm{signum}(x)$ for numeric input.
CSIGN	CPX h X.FCN CSIGN	DECM	(1) Returns the unit vector of $x + i\,y$ in **X** and **Y**.
SIGNMT	h MODE SIGNMT	All	(0) Sets sign-and-mantissa mode for integers. See p. 53ff.
SIN	f SIN	DECM	(1) Returns the sine of the angle in **X**.
SINC	h X.FCN SINC	DECM	(1) Returns $\frac{\sin(x)}{x}$; returns 1 for $x = 0$.
SINH	f HYP SIN	DECM	(1) Returns the hyperbolic sine of x.
SKIP	h P.FCN SKIP ***n***	¬INPUT	(0) Skips ***n*** program steps forwards ($0 \le n \le 255$). So e.g. SKIP 2 skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. **ATTENTION:** If you edit a section of your routine that is crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.
SL	h X.FCN SL ***n***	Integer	(1) Works like ***n*** (≤ 63) consecutive SLs on the *HP-16C*. SL 0 executes as NOP, but loads **L**. See p. 56 for more.
SLOW	h MODE SLOW	All	(0) Sets the processor speed to 'slow'. This is also automatically entered for low battery voltage (see p. 36). Compare FAST.
SLV	f SLV ***label***	DECM	Solves the equation $f(x) = 0$, with $f(x)$ calculated by the routine specified. Two initial estimates of the root must be supplied in **X** and **Y** when calling SLV. For the rest, the user interface is as in the *HP-15C*. This means SLV returns x_{root} in **X**, the second last x-value tested in **Y**, and $f(x_{root})$ in **Z**. It also means SLV acts as a test, so the next program step will be skipped if SLV fails. Please refer to the *HP-15C Owner's Handbook* (Section 13 and Appendix D) for more information about automatic root finding.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
SLVQ	h X.FCN SLVQ	DECM	Solves the quadratic equation $ax^2+bx+c=0$, with its real parameters on the input stack [***c, b, a***, …] , and tests the result. • If $r:=b^2-4ac\geq 0$, SLVQ returns $-\frac{b\pm\sqrt{r}}{2a}$ in **Y** and **X**. In a program, the step after SLVQ will be executed. • Else, SLVQ returns the real part of the first complex root in **X** and its imaginary part in **Y** (the 2nd root is the complex conjugate of the first – see CONJ). If run directly from the keyboard, the complex indicator C is lit then – in a program, the step after SLVQ will be skipped. In either case, SLVQ returns ***r*** in **Z**. Higher stack levels are kept unchanged. **L** will contain equation parameter ***c***.
SMODE?	h P.FCN SMODE?	INPUT	(-1) Returns the integer sign mode set, i.e. 2 (meaning 'true') for 2's complement, 1 ('true' again) for 1's complement, 0 (i.e. 'false') for unsigned, or -1 (i.e. 'true') for sign and mantissa mode.
SPEC?	h TEST SPEC?	INPUT	(0) True if ***x*** is 'special', i.e. infinite or non-numeric.
SR	h X.FCN SR ***n***	Integer	(1) Works like ***n*** (≤ 63) consecutive SRs on the *HP-16C*. SR 0 executes as NOP, but loads **L**. See p. 56 for more.
sRCL	h X.FCN sRCL ***s***	INPUT	(-1) Assumes the source contains *SP* data and recalls them as such. See pp. 159 and 186.
SSIZE4	h MODE SSIZE4	All	Set the stack size to 4 or 8 levels, respectively. See p. 17. Note register contents will remain unchanged in this operation. The same will happen if stack size is modified by any other operation (e.g. by RCLM, see p. 37).
SSIZE8	h MODE SSIZE8		
SSIZE?	h P.FCN SSIZE?	INPUT	(-1) Returns the number of stack levels allocated.
	h STAT	DECM	Catalog. See p. 125.
	h STATUS	INPUT	Browser. See p. 125.
STO	STO ***s***	INPUT	(0) See p. 33 for CSTO.
STOM	STO MODE ***s***	INPUT	(0) Stores mode settings for later use as described on p. 37 (no need pressing h here). RCLM recalls such data.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
STOP	R/S	STO	(0) Stops program execution. May be used to wait for input, for example.
STOPW	CPX R/S h X.FCN STOPW	DECM	Stopwatch application based on the real time clock and following the timer of the *HP-55*. See p. 145 for a detailed description. **ATTENTION**: This command will be part of the function set only if the proper firmware is loaded. See *Appendix A*.
STOS	h P.FCN STOS *s*	¬INPUT	(0) Stores all current stack levels in a set of 4 or 8 registers, starting at destination address *s*. See RCLS.
STO+	STO + *s*	¬INPUT	(0) Executes the specified operation on *s* and stores the result into said address. See p. 33 for ^{C}STO... E.g. STO–12 subtracts *x* from *r12* like the key-strokes RCL 12 x⇄y − STO 12 would do, but the stack remains unchanged.
STO–	STO − *s*		
STO×	STO × *s*		
STO/	STO / *s*		
STO↑	STO ▲ *s*	¬INPUT	(0) STO↑ (↓) takes the maximum (minimum) of *s* and *x* and stores it at the address specified.
STO↓	STO ▼ *s*		
SUM	h STAT SUM	DECM	(-2) Recalls the linear sums ***Σy*** and ***Σx***. Useful for elementary vector algebra in 2D.
	h SUMS	DECM	Catalog. See p. 125.
s_w	h STAT s_w	DECM	(-1) Calculates the standard deviation for weighted data (where the weight *y* of each data point *x* was entered via Σ+). See *Appendix I* for the formula.
s_{XY}	h STAT s_{XY}	DECM	(-1) Calculates the sample covariance for the two data sets entered via Σ+. It depends on the fit model selected. See *Appendix I* for the formula See COV for the population covariance.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
TAN	f TAN	DECM	(1) Returns the tangent of the angle in **X**.[57]
TANH	f HYP TAN	DECM	(1) Returns the hyperbolic tangent of x.
	h TEST	DECM	Catalog. See p. 125.
TICKS	h P.FCN TICKS	¬INPUT	(-1) Returns the number of ticks from the real time clock at execution time. With the quartz crystal built-in, 1 *tick* = 0.1 *s*. Without, it may be some 10% more or less. So the quartz crystal is inevitably required if the clock is to be useful in long range. TICKS also works without the quartz, however.
TIME	h X.FCN TIME	DECM	(-1) Recalls the time from the real time clock at execution, displaying it in the format `hh.mmss` in standard 24h-mode. Choose FIX 4 for best results.
T_n	h X.FCN T_n	DECM	(2) Chebychev polynomials of first kind. See *Appendix I* for details.
TOP?	h TEST TOP?	STO	(0) Returns `false` if TOP? is called in a subroutine, `true` if the program-running flag is set and the subroutine return stack pointer is clear.
TRANSP	h MATRIX TRANSP	DECM	(1) Takes a matrix *descriptor* x and returns the descriptor of its transpose. The transpose is done in-situ and does not require any additional memory.
$t_P(x)$	h PROB $t_P(x)$	DECM	(1) Student's t distribution. The degrees of freedom are stored in **J**. $t_u(x)$ equals Q(t) and $t^{-1}(p)$ equals t_P in the *HP-21S*. See p. 49 for an application example and *Appendix I* for more mathematical details.
$t_u(x)$	h PROB $t_u(x)$		
t(x)	h PROB t(x)		
$t^{-1}(p)$	h PROB $t^{-1}(p)$		
t⇆	h P.FCN t⇆ s	¬INPUT	Swaps t and s, in analogy to x⇆ .
ULP	h X.FCN ULP	¬INPUT	(1) Returns 1 times the smallest power of ten which can be added to x or subtracted from x to actually change the value of x in your *WP 34S* in the mode set. Thus 1 is returned in integer mode.
U_n	h X.FCN U_n	DECM	(2) Chebychev polynomials of second kind. See *Appendix I* for details.
UNSIGN	h MODE UNSIGN	All	(0) Sets unsigned mode like UNSGN does on the *HP-16C*.

[57] Note that TAN returns "not numeric" for $x = \pm 90°$ or equivalents if flag D is set.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
VERS	h X.FCN VERS	INPUT	(0) Shows your firmware version and build number.
VIEW	h VIEW *s*	INPUT	(0) Shows *s* until the next key is pressed. See p. 69 for more.
VIEWα	h P.FCN VIEWα	INPUT	(0) Displays ***alpha*** in the top row and - - - in the bottom row until next key is pressed (compare to AVIEW in the *HP-42S*). See p. 69 for more.
	h VIEW −	INPUT	
VWα+	h VIEW *s*	INPUT	(0) Displays ***alpha*** in the top row and *s* in the bottom row until the next key is pressed. See p. 69 for more.
	h P.FCN VWα+	INPUT	
WHO	h X.FCN WHO	INPUT	(0) Displays credits to the brave men who made this project work.
WDAY	h X.FCN WDAY	DECM	(1) Takes ***x*** as a date in the format selected, returns the name of the day in the dot matrix and a corresponding integer in the numeric display (Monday = 1, Sunday = 7) [58].
W_m	h X.FCN W_m	DECM	(1) W_p returns the principal branch of Lambert's W for given $x \geq -1/e$. W_m returns its negative branch. See Appendix I for more.
W_p	h X.FCN W_p		
W^{-1}	h X.FCN W^{-1}	DECM	(1) Returns ***x*** for given W_p (≥ -1). See there.
Weibl	h PROB Weibl	DECM	(1) Weibull distribution with its shape parameter ***b*** in **J** and its characteristic lifetime ***T*** in **K**. See Appendix I for details. Weibl^{-1} returns the survival time t_s for a given probability ***p*** in **X**, with ***b*** in **J** and ***T*** in **K**.
$Weibl_P$	h PROB $Weibl_P$		
$Weibl_u$	h PROB $Weibl_u$		
Weibl $^{-1}$	h PROB $Weibl^{-1}$		
WSIZE	h MODE WSIZE *n*	All (but becomes effective in integer modes only)	(0) Works almost like on the *HP-16C*, but with the parameter $1 \leq n \leq 64$ trailing the command instead of taken from **X**. Reducing the word size truncates the values in the stack registers employed and in **L**. All other memory content stays as is (see Appendix H for details). Increasing the word size will add empty bits to each stack level. WSIZE 0 sets the word size to maximum, i.e. 64 bits.
WSIZE?	h P.FCN WSIZE?	INPUT	(-1) Recalls the word size set.

[58] These numbers correspond to Chinese weekdays 1 to 6 directly. For Portuguese weekdays ('segunda feira' etc.), add 1 to days 1 to 5.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
x^2	g x^2	¬INPUT	(1)
x^3	h X.FCN x^3	¬INPUT	(1)
XEQ	XEQ *label*	STO	(0) Calls the subroutine with the label specified.
		¬STO, ¬INPUT	(0) Executes the program with the label specified.
	A , B , C , or D (you may need f for reaching these hotkeys in integer bases >10.)	STO	(0) Calls the respective subroutine, so e.g. XEQ C will be inserted when C is pressed.
		¬STO, ¬INPUT	(0) Executes the respective program if defined.
XEQα	h P.FCN XEQα	¬INPUT	(0) Takes the first three characters of ***alpha*** (or all if there are fewer than three) as a label and calls or executes the respective routine.
XNOR	h X.FCN XNOR	¬INPUT	(2) Works in analogy to AND. See *there*.
XOR	h XOR	¬INPUT	(2) Works in analogy to AND. See *there*.
XTAL?	h TEST XTAL?	¬INPUT	(0) Tests for presence of the crystal necessary for a precise real time clock (think of Xmas).
$\bar{x}$	f $\bar{x}$	DECM	(-2) Returns the arithmetic means of the x- and y-data accumulated. See also s, SERR, and σ .
$\bar{x}g$	h STAT $\bar{x}g$	DECM	(-2) Returns the geometric means of the data accumulated. See *Appendix I* for the formula. See also ε , ε_m , and ε_P .
$\bar{x}_w$	h STAT $\bar{x}_w$	DECM	(-1) Returns the arithmetic mean for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See *Appendix I* for the formula. See also s_w and $SERR_w$.
$\hat{x}$	h STAT $\hat{x}$	DECM	(1) Returns a forecast ***x*** for a given ***y*** (in **X**) following the fit model chosen. See L.R. for more.
	h X.FCN	All	Catalog. See p. *125*.
x !	h !	DECM	(1) Returns $\Gamma(x + 1)$.
		Integer	(1) Returns the factorial $n!$.
x → α	h X.FCN x→α	All	(0) Interprets ***x*** as a character code. Appends the respective character to ***alpha***, similar to XTOA in the *HP-42S*.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
$x\leftrightarrows$	h x⇄ s	¬INPUT	Swaps x and s, in analogy to $x\leftrightarrows y$. See p. 33 for ${}^{c}x\leftrightarrows$. Will be listed like x⇄ J, x⇄.12, x⇄→12, etc.
x⇄ Y	x⇄y	¬INPUT	Swaps the stack contents x and y, performing Re⇄Im if a complex operation was executed immediately before.
${}^{c}x\leftrightarrows$ Z	CPX x⇄y	¬INPUT	Swaps the complex stack contents x_c and y_c . See p. 31 for details.
x < ?	h TEST x<_? s	¬INPUT	(0) Compare x with s. E.g. h TEST x< ? K compares x with k , and will be listed as x<? K in a program. See the examples on p. 26 for more. x ≈ ? will be true if the rounded values of x and s are equal (see ROUND). The signed tests x = + 0 ? and x = – 0 ? are meant for integer modes 1COMPL and SIGNMT, and for DECM if flag **D** is set. In all these cases, e.g. 0 divided by –7 will display –0. CPX f x = ? s and CPX g x ≠ ? s compare the complex number $x + i\,y$ as explained on p. 33.
x ≤ ?	h TEST x≤_? s		
$x =$?	f x = ? s		
x = +0 ?	h TEST x=+0?		
x = –0 ?	h TEST x=–0?		
x ≈ ?	h TEST x≈_? s		
$x \neq$?	g x ≠ ? s		
x ≥ ?	h TEST x≥_? s		
x > ?	h TEST x>_? s		
$\sqrt[x]{y}$	h X.FCN x√y	¬INPUT	(2)
YEAR	h X.FCN YEAR	DECM	(1) Assumes x containing a date in the format selected and extracts the year.
y^x	f y^x	¬INPUT	(2) In integer modes, x must be ≥ 0.
	C	¬(INPUT, -3, -4, -5, h)	(2) Shortcut working if label C is not defined.
ŷ	f ŷ	DECM	(1) Returns a forecast y (in X) for a given x following the fit model chosen. See L.R. for more.
Y.MD	h MODE Y.MD	All	(0) Sets the format for date display.
y⇄	h P.FCN y⇄ s	¬INPUT	Swaps y and s, in analogy to $x\leftrightarrows$.
z⇄	h P.FCN z⇄ s	¬INPUT	Swaps z and s, in analogy to $x\leftrightarrows$.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
α	f α	STO & ¬INPUT	Turns on alpha mode for keyboard entry of alpha constants. **INPUT** is set and the previous program step stays displayed until a character is entered. Each such character (e.g. '?') will be stored in one program step (such as α? here) and will be appended to ***alpha*** in program execution. To start a new string, use CLα first.
		STO & INPUT	Turns on alpha group mode for direct entry of up to three characters in one program step taking two *words*. Your *WP 34S* will display α' in the top line. Now enter the characters you want to append to ***alpha***. To start a new string, use CLα before. **Example**: Entering f α T f ↑ E S f α T h PSE 1 will result in two program steps stored: α'Tes' α't 1' and Test 1 appended to ***alpha*** in program execution.[59]
		¬STO, ¬INPUT	Enters alpha mode for appending characters to ***alpha***. To start a new string, use CLα first.
		¬STO & INPUT	Leaves alpha mode.
αDATE	h X.FCN αDATE	¬integer	(0) Takes ***x*** as a date and appends it to ***alpha*** in the format set. See DATE. To append a date stamp to ***alpha***, call DATE αDATE. For a short European date stamp, set FIX 2, RDX. and call DATE αRC# X.
αDAY	h X.FCN αDAY	¬integer	(0) Takes ***x*** as a date, recalls the name of the respective day and appends its first three letters to ***alpha***.
αGTO	h P.FCN αGTO s	¬INPUT	(0) Interprets ***s*** as character code. Takes the first three characters of the converted code (or all if there are fewer than three) as an alpha label and positions the program pointer to it.
αIP	h X.FCN αIP	All	(0) Appends the integer part of ***x*** to ***alpha***, similar to AIP in the *HP-42S*.

[59] Note alpha group mode is left automatically after three characters are put in, so it must be called again for continuation. Some characters must not be entered at third position (see Appendix E).

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
αLENG	h X.FCN αLENG	All	(-1) Returns the number of characters found in ***alpha***, like ALENG in the *HP-42S*.
αMONTH	h X.FCN αMONTH	¬integer	(0) Takes ***x*** as a date, recalls the name of the respective month and appends its first 3 letters to ***alpha***.
αOFF	h P.FCN αOFF	¬INPUT	(0) Work like AOFF and AON in the *HP-42S*, turning alpha mode off and on.
αON	h P.FCN αON		
αRCL	f RCL ***s***	INPUT	(0) Interprets ***s*** as characters and appends them to ***alpha***.
	h X.FCN αRCL ***s***	¬INPUT	
αRC#	h X.FCN αRC# ***s***	All	(0) Interprets ***s*** as a number, converts it to a string in the format set, and appends this to ***alpha***. **Example**: If ***s*** is 1234 and ENG 2 and RDX. are set, then 1.23e3 will be appended. See also αDATE for an application.
αRL	h X.FCN αRL ***n***	All	(0) Rotates ***alpha*** by ***n*** characters like AROT in the *HP-42S*, but with ***n*** ≥ *0* and the parameter trailing the command instead of taken from **X**. αRL 0 executes as NOP, but loads **L**.
αRR	h X.FCN αRR ***n***	All	(0) Works like αRL but rotates to the right.
αSL	h X.FCN αSL ***n***	All	(0) Shifts the ***n*** leftmost characters out of ***alpha***, like ASHF in the *HP-42S*. αSL 0 equals NOP, but loads **L**.
αSR	h X.FCN αSR ***n***	All	(0) Works like αSL but takes the ***s*** rightmost characters instead.
αSTO	f STO ***s***	INPUT	(0) Stores the first (i.e. leftmost) 6 characters of ***alpha*** in the destination specified. Stores all if there are fewer than six.
	h X.FCN αSTO ***s***	¬INPUT	
αTIME	h X.FCN αTIME	¬integer	(0) Takes ***x*** as a decimal time and appends it to ***alpha*** in the format `hh:mm:ss` according to the time mode selected. See 12h, 24h, and TIME. To append a time stamp to ***alpha***, call TIME αTIME.
αXEQ	h P.FCN αXEQ ***s***	¬INPUT	(0) Interprets ***s*** as character code. Takes the first three characters (or all if there are fewer than three) of the converted code as an alpha label and calls or executes the respective routine.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
α → x	h X.FCN α→x	All	(-1) Returns the character code of the leftmost character in ***alpha*** and removes this character from ***alpha***, like ATOX in the *HP-42S*.
β	h X.FCN β	DECM	(2) Returns *Euler's Beta* $B(x,y)=\frac{\Gamma(x)\cdot\Gamma(y)}{\Gamma(x+y)}$ with $\mathrm{Re}(x)>0$, $\mathrm{Re}(y)>0$. Called β here to avoid ambiguity.
Γ	h X.FCN Γ	DECM	(1) Returns $\Gamma(x)$. Additionally, h ! calls $\Gamma(x+1)$. See also LNΓ.
γ_{XY}	h X.FCN γXY	DECM	(2) Returns the lower or upper incomplete gamma function, respectively. See *Appendix I* for details.
Γ_{XY}	h X.FCN ΓXY		
ΔDAYS	h X.FCN ΔDAYS	DECM	(2) Assumes **X** and **Y** containing dates in the format chosen and calculates the number of days between them like in the *HP-12C*.
Δ%	g Δ%	DECM	(1) Returns $100\cdot\frac{x-y}{y}$ like %CH in the *HP-42S*.
ε	h STAT ε	DECM	(-2) Returns the scattering factors ε_y and ε_x for log-normally distributed sample data. This ε_x works for the geometric mean $\bar{x}_g$ in analogy to the standard deviation s for the arithmetic mean x̄ but multiplicative instead of additive. See *Appendix I* for more.
ε_m	h STAT εm	DECM	(-2) Works like ε but returns the scattering factors of the two geometric means.
ε_P	h STAT εP	DECM	(-2) Works like ε but returns the scattering factors of the two populations.
ζ	h X.FCN ζ	DECM	(1) Returns Riemann's Zeta. See *App. I* for more.
π	h π	DECM	(-1) Recalls **π**.
$^{c}\pi$	CPX π	DECM	(-2) Recalls **π** into **X** and clears **Y**.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Π	f π ***label***	DECM	(1) Computes a product using the routine specified. Initially, **X** contains the loop control number in the format `cccccc.fffii`, and the product is set to 1. Each run through the routine specified by ***s*** computes a factor. At its end, this factor is multiplied with said product; the operation then decrements `cccccc` by `ii` and runs said routine again if then `cccccc` ≥ `fff`, else returns the resulting product in **X**.
σ	h STAT σ	DECM	(-2) Works like s but returns the standard deviations of the two populations instead. See *Appendix I* for the formula.
Σ	g Σ ***label***	DECM	(1) Computes a sum using the routine specified. Initially, **X** contains the loop control number in the format `cccccc.fffii`, and the sum is set to 0. Each run through the routine specified by ***s*** computes a summand. At its end, this summand is added to said sum; the operation then decrements `cccccc` by `ii` and runs said routine again if then `cccccc` ≥ `fff`, else returns the resulting sum.
$\Sigma \ln^2 x$	h SUMS Σln²x etc.	DECM	(-1) Recall the respective statistical sums. These sums are necessary for curve fitting models beyond pure linear. Calling them by name significantly improves program readability. Note these sums are stored in special registers in your *WP 34S*. **ATTENTION:** Depending on input data, some or all of these sums may become non-numeric.
$\Sigma \ln^2 y$			
$\Sigma \ln x$			
$\Sigma \ln xy$			
$\Sigma \ln y$			
$\Sigma x \ln y$			
$\Sigma y \ln x$			
σ_w	h STAT σ_w	DECM	(-1) Works like s_w but returns the standard deviation of the population instead. See *Appendix I* for the formula.
Σx	h SUMS Σx etc.	DECM	(-1) Recall the respective statistical sums. These sums are necessary for basic statistics and linear curve fitting. Calling them by name significantly improves program readability. Note these sums are stored in special registers of your *WP 34S*.
Σx^2			
$\Sigma x^2 y$			
Σxy			
Σy			
Σy^2			

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
Σ+	h Σ+	DECM	Adds a data point to the statistical sums. Both functions disable stack lift as usual. Both may be used for 2D vector adding and subtracting as well.
	A	DECM	Shortcut working if label A is not defined.
Σ–	h Σ–	DECM	Subtracts a data point from the statistical sums.
$\Phi_u(x)$	h PROB Φu(x)	DECM	(1) Standard normal *error probability*, equaling Q in the *HP-32E* and Q(z) in the *HP-21S*.
$\varphi(x)$	h PROB Φ(x)	DECM	(1) Standard normal *pdf*. See *Appendix I* for the formula.
Φ(x)	f Φ	DECM	(1) Standard normal *cdf*.
Φ^{-1}(p)	g Φ^{-1}	DECM	(1) Equals Q^{-1} in the *HP-32E* and z_p in the *HP-21S*.
χ^2	h PROB χ²	DECM	(1) Chi square distribution. The *cdf* χ^2 (with its degrees of freedom given in **J**) equals $1 - Q(\chi^2)$, χ^2_u equals $Q(\chi^2)$ and χ^2INV equals χ^2_p in the *HP-21S*. See *Appendix I* for more.
χ^2INV	h PROB χ²INV		
χ^2_P	h PROB χ²P		
χ^2_u	h PROB χ²u		
(-1)^x	h X.FCN (–1)ˣ	¬INPUT	(1) If x is not a natural number, returns $\cos(\pi \cdot x)$.
+	+	¬INPUT	(2) Returns $y + x$.
	CPX +	DECM	(2) Returns $[x+z,\ y+t,\ \ldots]$. May be used for adding 2D vectors as well.
–	–	¬INPUT	(2) Returns $y - x$.
	CPX –	DECM	(2) Returns $[x-z,\ y-t,\ \ldots]$. May be used for subtracting 2D vectors as well.
×	×	¬INPUT	(2) Returns $y \cdot x$.
	CPX ×	DECM	(2) Returns $[x \cdot z - y \cdot t,\ x \cdot t + z \cdot y,\ \ldots]$. Look at CROSS or DOT for multiplying 2D vectors.
×MOD	h X.FCN ×MOD	Integer	(3) Returns $(z \cdot y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
/	[/]	DECM	(2) Returns y / x .
		Integer	(2) Returns y div x . Compare IDIV.
	[CPX] [/]	DECM	(2) Returns $\left[\frac{x \cdot z + y \cdot t}{z^2+t^2}, \frac{z \cdot y - x \cdot t}{z^2+t^2}, \ldots\right]$.
+/–	[+/–]	INPUT	(1) 'Unary minus', corresponding to $x \cdot (-1)$ or $x_c \cdot (-1)$, respectively.
→DATE	[h] [X.FCN] →DATE	DECM	(3) Assumes the three components of a date (year, month, and day) supplied on the stack in proper order for the date format selected and converts them to a single date in x. Thus inverts DATE→.
→DEG	[→] [DEG]	DECM	(1) Takes x as an angle in the angular mode currently set and converts it to *degrees*. Prefix [g] may be omitted.
→GRAD	[→] [GRAD]	DECM	(1) Works like →DEG, but converts to *gon* (*grad*).
→H	[→] [f] [H.d]	DECM	(1) Takes x as *hours* or *degrees* in the format `hhhh.mmssdd` as in vintage *HP* calculators and converts it to a decimal time or angle, allowing for using standard arithmetic operations then.
→H.MS	[→] [f] [H.MS]	DECM	(1) Takes x as decimal *hours* or *degrees* and converts it to the format `hhhh.mmssdd` as in vintage *HP* calculators. For calculations, use H.MS+ or H.MS– then.
→POL	[g] [→P]	DECM	Assumes **X** and **Y** containing 2D Cartesian coordinates (x, y) of a point or components of a vector and converts them to the polar coordinates / components (r, ϑ) with the radius $r = \sqrt{x^2 + y^2}$
→RAD	[→] [RAD]	DECM	(1) Works like →DEG, but converts to *radians*.
→REC	[f] [R←]	DECM	Assumes **X** and **Y** containing 2D polar coordinates (r, ϑ) of a point or components of a vector and converts them to the Cartesian coordinates or components (x, y) .

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
⇆	h P.FCN ⇆____	¬INPUT	Shuffles the contents of the bottom four stack levels at execution time. **Examples**: ⇆XXYZ works like ENTER↑ (but does not disable stack lift!), ⇆YZTX works like R↓, ⇆ZTXY works like ^{C}x⇆y, but ⇆ZZZX is possible as well. **ATTENTION:** This is a very powerful command though it does not look it. Remember it will not affect the higher levels in an 8-level stack. If you play with this command, you may lose some level contents and can easily make a mess of the stack.
%	f %	DECM	(1) Returns $\frac{x \cdot y}{100}$, leaving **Y** unchanged.
%MG	h X.FCN %MG	DECM	(2) Returns the margin[60] $100 \cdot \frac{x-y}{x}$ in % for a price x and cost y, like %MU-Price in the *HP-17B*.
%MRR	h X.FCN %MRR	DECM	(3) Returns the mean rate of return in percent per period, i.e. $100 \cdot \left[\left(\frac{x}{y}\right)^{1/z} - 1\right]$ with x = future value after z periods, y = present value. For $z = 1$, Δ% returns the same result easier.
%T	h X.FCN %T	DECM	(2) Returns $100 \cdot \frac{x}{y}$, interpreted as % of total.
%Σ	h X.FCN %Σ h STAT %Σ	DECM	(1) Returns $100 \cdot \frac{x}{\sum x}$.
%+MG	h X.FCN %+MG	DECM	(2) Calculates a sales price by adding a margin of x % to the cost y, as %MU-Price does in the *HP-17B*. Formula: $p_{sale} = \frac{y}{1 - \frac{x}{100}}$ You may use %+MG for calculating net amounts as well. Just enter a negative percentage in x. **Example**: Total billed = 221,82 €, VAT = 19%. What is the net? 221,82 ENTER↑ 19 +/- X.FCN ▲ returns 186,40.

[60] Margin translates to „Handelsspanne“ in German.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
√	f √x	¬INPUT	(1)
	D	¬(INPUT, ¬4, ¬5, h)	(1) Shortcut working if label D is not defined.
∫	g ∫ *label*	DECM	Integrates the function given in the routine specified. Lower and upper integration limits must be supplied in **Y** and **X**, respectively. Otherwise, the user interface is as in the *HP-15C*. Please turn to the *HP-15C Owner's Handbook* (Section 14 and Appendix E) for more information about automatic integration and some caveats.
∞?	h TEST ∞?	¬INPUT	(0) Tests x for infinity.
^MOD	h X.FCN ^MOD	Integer	(3) Returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. **Example**: 10 73 ENTER↑ 55 ENTER↑ 31 X.FCN ▲ XEQ returns 26.
//	g //	DECM	(2) Returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{1}$, being very useful in electrical engineering especially. Returns 0 for x or y being zero.
🖶ADV	h P.FCN 🖶ADV	¬INPUT	(0) Prints the current contents of the print buffer plus a linefeed. **ATTENTION:** Any printing will only work with an hardware modification (see *Appendix H*) or using the *WP 34S* emulator in combination with a printer emulator (see *Appendix D*). Else print commands will be ignored. – The printer will actually print only when a line feed is sent to it.
🖶CHR	h P.FCN 🖶CHR *n*	¬INPUT	(0) Sends a single character (with the code specified) to the printer. Character codes $n > 127$ can only be specified indirectly. 🖶MODE setting will be honored. See 🖶ADV.
🖶PLOT	h P.FCN 🖶PLOT *s*	¬INPUT	(-1) Sends the graphic block starting at address s to the printer. If its width is 166, the data will be trailed by a line feed. See 🖶ADV and gDIM.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
$\text{🖶}^{C}r_{XY}$	h P.FCN 🖶ᶜrxy *s*	INPUT	(0) Prints the register specified and the next one, i.e. prints an entire complex number. A semicolon will separate both components. Works like 🖶r otherwise. **Example**: Assume 🖶MODE 1, SCI 1, $x = -1.2$ and $y = 0.34$. Then the output of 🖶ᶜrxy X would look like −1.2e0 ; 3.4e−1
🖶DLAY	h MODE 🖶DLAY *n*	All	(0) Sets a delay of s ticks (see TICKS) to be used with each line feed on the printer.
🖶MODE	h MODE 🖶MODE *n*	All	(0) Sets print mode. Legal print modes are: 0: Use the printer font and character set wherever possible (default). All characters feature the same width (5 columns + 2 columns spacing). 1: Use the variable pitch display font, resulting in some jitter on the printout but packing more characters in a line. 2: Use the small display font, which allows for packing even more info in a line. 3: Send the output to the serial channel. Works for plain ASCII only – no characters will be translated. Line setup is the same as for serial communication: 9600 *baud*, 8 *bits*, no parity.
🖶PROG	h P.FCN 🖶PROG	INPUT	(0) Prints the listing of the current program, one line per step. The current program is the one the program pointer is in at execution time. See 🖶ADV.
🖶r	h P.FCN 🖶r *s*	INPUT	(0) Prints the register specified, right adjusted, without labeling the output. If you want a heading label, call 🖶α+ first or use 🖶REGS. See 🖶ADV.
🖶REGS	h P.FCN 🖶REGS	INPUT	(1) Interprets x in the form `sss.nn` . Prints the contents of ***nn*** registers starting with number ***sss*** . Each register takes one line starting with a label. **ATTENTION for *nn* = 0 :** For $sss \in [0;99]$, printing will stop at the highest allocated global numbered register. For $sss \in [100;111]$, printing will stop at **K**. For $sss \geq 112$, printing will stop at the highest allocated local register. See also 🖶ADV.
🖶STK	h P.FCN 🖶STK	INPUT	(0) Prints the stack contents. Each level prints in a separate line starting with a label. See 🖶ADV.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
⎙TAB	h P.FCN ⎙TAB *n*	¬INPUT	(0) Positions the print head to print column ***n*** (0 to 165, where ***n*** > 127 can only be specified indirectly). Useful for formatting (in ⎙MODE 1 or 2 in particular). Allows also for printer plots. If ***n*** is less than current position, a linefeed will be entered to reach the new position. See ⎙ADV.
⎙WIDTH	h P.FCN ⎙WIDTH	¬INPUT	(-1) Returns the number of print columns ***alpha*** would take in the print mode set. See ⎙ADV and ⎙MODE. Second use: in ⎙MODE 1 or 2, ⎙WIDTH returns the width of ***alpha*** in pixels (including the last column being always blank) in the specified font.
⎙α	h P.FCN ⎙α	¬INPUT	(0) Appends ***alpha*** to the print line, trailed by a line feed. Compare ⎙α+ and ⎙+α. See ⎙ADV.
⎙α+	h P.FCN ⎙α+	¬INPUT	(0) Sends ***alpha*** to the printer without a trailing line feed, allowing further information to be appended to this line. May be repeated. See also ⎙ADV, ⎙r and ⎙+α.
⎙Σ	h P.FCN ⎙Σ	¬INPUT	(0) Prints the summation registers. Each register prints in one line starting with a label. See ⎙ADV.
⎙+α	h P.FCN ⎙+α	¬INPUT	(0) Appends ***alpha*** to the print line, adjusted to the right and trailed by a line feed. Compare ⎙α and ⎙α+. See ⎙ADV. **Example:** The following program section CLα α'Lef' α t ⎙α+ CLα α'Rig' α'ht' ⎙+α will print, if ⎙MODE 1 is set: Left Right
⎙?	h TEST ⎙?	¬INPUT	(0) Tests if the quartz crystal and the necessary firmware are installed for printing.
⎙#	h P.FCN ⎙# *n*	¬INPUT	(0) Sends a single *byte*, without translation, to the printer (e.g. a control code). ***n*** > 127 can only be specified indirectly. ⎙MODE setting will not be honored. See ⎙ADV.
#	h CONST # *n*	STO	Inserts an integer $0 \le n \le 255$ in a single step, thus saving up to two steps and an ENTER.

Name	Keys to press	in modes	Remarks (see p. 74 for general information)
$^{C}\#$	CPX h CONST # *n*	DECM	Works like # but also clears *y*. The shortcut works only for $1 \le n \le 9$.
	CPX *n*		

Nonprogrammable Control, Clearing and Information Keys

Besides the commands CLALL, CLPALL, GTO., LOAD, LOADP, PSTO, RESET, and SAVE, the following keystrokes cannot be programmed under the conditions stated:

Key(s) pressed	in modes	Remarks
f 1/x	In CONV	Inverts the current conversion (see p. 139).
8	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other response except R/S = **Y** , EXIT or ← for 'no' will be ignored.
	In a browser	Will be honored as described in the respective sections. See the note below this list.
ENTER↑ [61]	CAT, SHOW, STOPW	Will be honored as described in the respective sections. See the note below this list.
	÷	Selects the current item like XEQ below.
	INPUT	Leaves alpha mode.
	Else	Calls the programmable command ENTER described in the *IOP*.

[61] The mode conditions specified here will be checked top down for this command at execution time:

If CAT or STOPW is open, it will be served as described there;
else if there is an open catalog, the current item will be selected;
else if alpha mode is set, it will be left;
else the programmable command ENTER will be executed or inserted.

This method holds for all commands listed here using this triangular symbolic.

Key(s) pressed	in modes	Remarks
EXIT	¬RPN	Clears the *temporary message* displayed (see p. 38) returning to the calculator state as was before that message was thrown.
	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other response except R/S = **Y** , 8 or ← for 'no' will be ignored.
	In STOPW	Leaves the application (see p. 145).
	In a catalog or browser	Leaves the current catalog or browser without executing anything, returning to the *WP 34S* status as it was before.
	Command input pending	Cancels the execution of the pending operation, returning to the *WP 34S* status as it was before.
	RCL	Stops the running program like R/S below.
	STO	Leaves programming mode like P/R below.
	INPUT	Leaves alpha mode like ENTER↑ above.
	Else	Does nothing.
ON	Calculator off	Turns your *WP 34S* on.
	Else	ON-key combinations are found in the appendices. Most important is ON + + or − for adjusting display contrast.
h P/R	¬INPUT	Toggles programming mode (STO).
R/S	Asking for confirmation	Answers the question Sure? with **Y** for 'yes'. Any other response except 8 or EXIT or ← for 'no' will be ignored.
	CAT/STOPW	Will be honored as described in the respective sections. See the note below this list.
	RCL	Stops program execution immediately. Stopped will be displayed until the next keystroke.
	¬(STO, INPUT)	Runs the current program or resumes its execution starting with the current step.
	INPUT	Appends a Y or y to *alpha*.
	STO	Enters the command STOP described in the *IOP*.
XEQ	In CAT	Will be honored as described in the respective section. See the note below this list.
	÷	Selects the item currently displayed and exits, executing the respective command.
	Else	Calls the programmable command XEQ described in the *IOP*.

Key(s) pressed	in modes	Remarks
⬅	¬RPN	Clears the *temporary message* displayed (see p. 38) returning to the calculator state as was before that message was thrown.
	Asking for confirmation	Answers the question Sure? with **N** for 'no'. Any other response except R/S = **Y** , 8 or EXIT for 'no' will be ignored.
	In STOPW	Resets the timer (see p. 145).
	In a catalog or browser	Leaves the catalog or browser like EXIT above.
	Command input pending	Deletes the last digit or character put in. If there is none yet, cancels the pending command like EXIT above.
	INPUT	Deletes the rightmost character in ***alpha***.
	STO	Deletes current program step.
	Else	Calls the programmable command CLx described in the *IOP*.
f ◀ / g ▶	Integer	Shifts the display window to the left / right like in the *HP-16C*. Helpful while working with small bases. See p. 53.
	DECM	Shows the full mantissa and the full exponent until the next key is pressed (see p. 40).[62] Compare SHOW in previous calculators.
→ f 2	DECM	Shows x as an integer to base 2, 8, 10, or 16, respectively. Returns to the base set with the next keystroke. Prefix g may be omitted here.
→ 8		
→ f 10		
→ 16		
f ⬆	÷	Calls the character ⌸.
	INPUT	Toggles upper and lower case (the latter is indicated by ⬇).
	Else	Calls the programmable command ⌸r X (see there).

[62] These keys act differently in *DP* mode as described in Appendix H *below*.

Key(s) pressed	in modes	Remarks
▲ / ▼	In a browser or STOPW	Will be honored as described in the respective sections. See the note below this list.
	÷	Goes to previous / next item in the current catalog.
	INPUT	Scrolls the display window six characters to the left/right in ***alpha*** if possible. If fewer than six characters are beyond the limits of the display window on this side, the window will be positioned to the beginning/end of string. Useful for longer strings.
	Else	Acts like the command BST / SST in the *HP-42S*. I.e. browses programs in programming mode, where ▲ / ▼ will repeat with 5*Hz* when held down for longer than 0.5*s*. – Out of programming mode, SST will also execute the current program step, but the keys will not repeat.

See the section about *catalogs and browsers* (beginning on p. *125*) to learn more about them. Note that the browsers CAT, SHOW, STATUS, and the application STOPW (see p. *145*f) use some of the keys above as well as some more for particular control purposes (e.g. **STO**, **RCL**, ., and some digit keys). Check the respective entries below.

Alphanumeric Input

Character	Keys to press	in modes	Remarks
␣	h PSE	**INPUT**	Appends a blank space to ***alpha***.
°	.	DECM	Separates *degrees* or *hours* from *minutes* and *seconds*, so input format is `hhhh.mmssdd`. The user has to take care where an arbitrary real number represents such an angle or time.
0 … 9	0 … 9	¬INPUT	Standard numeric input. For integer bases <10, input of illegal digits is blocked. Note you cannot enter more than 12 digits in the mantissa.
		in addressing	Register input. See the *tables* above for the number ranges.
	0, 1, f 2, … , f 9	**INPUT**	Appends the respective digit to ***alpha***.
A … F	A … F (grey print)	-1 -2 -3 -4 -5 h	Numeric input for digits >10. See pp. *52*ff for more.
A … Z	A … Z (grey print)	in addressing	Register input. See the *virtual keyboard* on p. *25* for the letters applicable.
		INPUT	Appends the respective Latin letter to ***alpha***. Use f ↑ to toggle between cases.
E	EEX	DECM & ¬FRC	Works like E in the Pioneers.
i	CPX .	DECM & ¬FRC	Enters complex number i , i.e. $x = 0$ and $y = 1$.
A … Ω	g A … g O (grey print)	**INPUT**	Appends the respective Greek letter to ***alpha***. Use f ↑ to toggle between cases. See p. *62* for more.
(	f ◀(	**INPUT**	Appends the respective symbol to ***alpha***.
)	g)▶		
+	f +		
–	f –		
x	f x		

Character	Keys to press	in modes	Remarks
/	Second [.]	DECM	A second [.] in input turns to fraction mode and is interpreted as explained below. Note you cannot enter [EEX] after you entered [.] twice – but you may delete the second dot while editing the input line.
		FRC	First [.] is interpreted as a space, second as a fraction mark. See p. 41 for some examples.
	[f] [/]	INPUT	Appends a slash to ***alpha***.
+/-	[+/-]	INPUT	Works like [+/-] in the Pioneers.
±	[f] [+/-]	INPUT	Appends the respective symbol to ***alpha***.
,	[h] [./,] [XEQ]		
.	[f] [.]		
'.' or ','	[.]	DECM	Inserts a radix mark as selected.
!	[h] [!]	INPUT	Appends the respective symbol to ***alpha***.
?	[h] [▼]		
⇆	[h] [x⇄]		
≠	[h] [XOR]		
&	[h] [AND]		
\	[h] [/]		
|	[h] [OR]	INPUT	Appends a vertical separator to ***alpha***.
🖶	[f] [↑]	In a catalog	Enters the print character for fast access to the respective commands (see p. 133).

CATALOGS AND BROWSERS

Due to the large set of operations your *WP 34S* features, most of them are stored in catalogs as was discussed from p. 22 on. Numeric constants (in CONST and CONV) or characters (in the alpha catalogs) are other items contained in catalogs. Opening a **catalog** on your *WP 34S* will set alpha mode to allow for typing the first character(s) of the item wanted for rapidly accessing it. A subset of the full alpha keyboard shown on p. 62 is sufficient for catalog browsing as pictured here. But there are three differences:

f B (= 1/x) is for reverting conversions easily (see p. 139).

f → just calls the character '→', and f EXIT calls the print character in catalog browsing (since case switching is not needed here).

▲ and ▼ will browse the open catalog.

ENTER↑ or XEQ select the item displayed, recall or execute it, and exit the catalog.

EXIT or ← leave the catalog without executing anything, i.e. they cancel the catalog call.

See p. 133 for some examples.

You may switch catalogs easily by just calling a new one accessible in current mode directly from the catalog you are browsing – no need to EXIT first.

When the last catalog called is reopened, the last item viewed therein is immediately displayed for easy repetitive use. A single function may be contained in more than one catalog.

There are also three ***browsers*** featured for checking memory, flags, program labels and registers (i.e. CAT, SHOW, and STATUS). They may be called in all modes except alpha. Therein, EXIT and ⇦ work as in catalogs. SHOW and STATUS operate in *α_T mode*, however. And some special keys and special rules may apply in browsers as explained in the following table:

Keys to press	Contents and special remarks
h CAT	Defined alphanumeric (i.e. global) labels. The first item displayed is the top global label of the *current program*[63] – if there is no such label, the end of this program is shown. ▲ and ▼ browse global labels, while the location of the label displayed is indicated in the lower line (rAM for RAM, Lib for the library in *FM*, or buP for the backup region therein). Duplicate labels also show the primary address, e.g. CALLS 012 when found a second time in RAM, or e.g. CALLS Lib when found a second time elsewhere. f ▲ and f ▼ browse programs, i.e. show the first label in previous or next program (with programs separated by END statements). 0, 1, and 2 allow quick jumps to the top of rAM, Lib, or buP, respectively. ENTER↑ goes to the alpha label displayed, while XEQ executes it. Both key-strokes will perform a label search as described on p. 67. In programming mode, ENTER↑ will insert a GTO and XEQ an XEQ with said label trailing. R/S starts the current program without performing a label search first. RCL and STO execute PRCL or PSTO for the current program, respectively. The latter makes sense if called in rAM or buP only. P/R enters mode **STO**, allowing for browsing or editing the current program. The latter will work in rAM only. Note that ⇦ will delete a step, while EXIT or another P/R will leave CAT immediately. CLP deletes the current program, be it in rAM or Lib.
g SHOW	Browses all allocated stack and general purpose registers showing their contents, starting with **X**. ▲ goes up the stack, continuing with the other lettered registers, then with **R00**, **R01**, etc. ▼ browses the registers going down from the highest allocated numbered register (**R99** in startup default) to **R00** if applicable, then continuing with **K, J**, etc. . turns to local registers if applicable, starting with **R.00**. Then, ▲ and ▼ browse local registers up and down until another . returns to **X**. Local register addresses may exceed .15 here! Input of any legal letter or two-digit number jumps to the corresponding register (see p. 26). ENTER↑ or RCL recall the register displayed. In programming mode, they enter a corresponding step RCL … (In your *WP 34S*, the keys ◀ and ▶ do what SHOW did in vintage calculators – see p. 40.)

[63] The *current program* is the one the program pointer is in, as mentioned above.

Keys to press	Contents and special remarks
h STATUS	Displays the memory status and browses all user flags, similar to STATUS on the *HP-16C*. It shows the amount of free memory *words* in *RAM* and *FM* first, e.g.: Free: 516 , FL. 9999 . Press ▼ and read if there are summation registers used, plus the number of global numbered registers and local registers allocated: Regs: 96 , Loc. 7 or Regs: Σ+ 96 , Loc. 7 . Another ▼ presents the status of the first 30 user flags in one very concise display as explained below. **Example**: If flags 2, 3, 5, 7, 11, 13, 14, 17, 19, 20, 26, and X are set, and labels B and D are defined in program memory, STATUS ▼ ▼ will display this: FL 00–29 bd Where the mantissa is usually displayed, there are now three rows of horizontal bars. Each row shows the status of 10 flags. If a particular flag is set, the corresponding bar is lit. So here the top row of bars indicates flags 0 and 1 are clear, 2 and 3 set, and 4 clear. Then a ‖ separates the first five flags from the next. The following top-row bars indicate flag 5 set, 6 clear, 7 set, 8 and 9 clear. The next two rows show the status up to flag 29. Pressing ▼ once again will increment the start address by ten; so the display will look like: FL 10–39 bd Another ▼ will show flags 20 - 49 etc. until 70 - 99, 80 - 99, 90 - 99. A final ▼ will display the last 12 global flags in rows of four – note flag X is shown being set as we expect: XYZTA:DLIJK bd The exponent section indicates the status of the four hotkeys – if all four labels are defined in programs then ALL will be shown there. ▲ browses backwards. Alternatively, pressing a digit (e.g. 5) will display up to 30 flags starting with 10 times this digit (flags 50 – 79 here). Pressing a legal letter such as D will display the top 12 flags.

Finally, see the following list of **catalogs** featured:

Keys to press	in modes	Contents and special remarks
h CONST	DECM & ¬STO	Constants like in the *HP-35s*, but more. See them listed on p. 134ff. While browsing this catalog, the values of the constants are displayed. Picking a constant will recall it.
	Integer	Calls the command # (see *the IOP*).
	DECM & STO	Picking a constant will insert a program step containing the name of the constant selected, preceded by # . This step will then recall the value of said constant in program execution.
CPX h CONST	DECM & ¬STO	Opens the same catalog of constants as h CONST, but picking a constant will execute a complex recall. So, a stack looking like $[x, y, \ldots]$ before will contain $[\textbf{\textit{constant}}, 0, x, y, \ldots]$ after picking.
	DECM & STO	Picking a constant will insert a program step containing the name of the constant selected, preceded by ⌜# . This step will then perform a complex recall of the value of said constant in program execution.
h CONV	DECM	Conversions as listed in a *table below*. While browsing this catalog, the converted content of **X** is displayed. Picking a conversion will return this value.
f CPX	INPUT	'Complex' letters mandatory for many languages (see p. 132). Case may be toggled here (see f ↑ above).
h MATRIX	DECM	Matrix operations library.
h MODE	¬INPUT	Mode setting functions.
h PROB	DECM	Probability distributions beyond the standard normal and its inverse.
h P.FCN	¬INPUT	Extra programming and I/O functions.
h R↑	INPUT	Superscripts and subscripts (see p. 132).
h STAT	DECM	Extra statistical functions.
h SUMS	DECM	All summation registers. While browsing this catalog, register contents are displayed. Picking a register will recall its contents.
h TEST	¬INPUT	All tests except the two on the keyboard (see p. 130).
	INPUT	Comparison symbols and brackets, except (,), and ≠ (see p. 132).

Keys to press	in modes	Contents and special remarks	
h X.FCN	DECM	Extra real functions.	These three catalogs are merged in mode STO to ease programming.
	Integer	Extra integer functions.	
	INPUT	Extra alpha functions.	
CPX h X.FCN	DECM	Extra complex functions.	
h ./,	INPUT	Punctuation marks and text symbols (see p. 132).	
f →	INPUT	Arrows and mathematical symbols (see p. 132).	

See the next pages for detailed contents of the various catalogs. Items are sorted alphabetically within the catalogs (see p. 74 for the sorting order). You may access a particular item quickly by typing the first characters of its name – see p. 133 for examples and constraints.

Note neither catalog nor browser calls can be programmed.

Catalog Contents in Detail

MATRIX
DET
LINEQS
MROW+×
MROW×
MROW⇆
M+×
M^{-1}
M-ALL
M-COL
M-DIAG
M-ROW
M×
M.COPY
M.IJ
M.LU
M.REG
nCOL
nROW
TRANSP

MODE
12h
1COMPL
24h
2COMPL
BASE
BestF
DBLOFF
DBLON
DENANY
DENFAC
DENFIX
DENMAX
DISP
D.MY
E3OFF
E3ON
ExpF
FAST
FRACT
JG1582
JG1752
LinF
LogF
LZOFF
LZON
M.DY
PowerF
RCLM
RDX,
RDX.
REGS
RMF
SEPOFF
SEPON
SETCHN
SETDAT
SETEUR
SETIND
SETJPN
SETTIM

PROB
Binom
$Binom_P$
$Binom_u$
$Binom^{-1}$
Cauch
…
Expon
…
$F_P(x)$
$F_u(x)$
F(x)
$F^{-1}(p)$
Geom
…
Lgnrm
…
Logis
…
Norml
…
Poiss
…
Poisλ
…
$t_P(x)$
$t_u(x)$
t(x)
$t^{-1}(p)$
Weibl
…
$\Phi_u(x)$
φ(x)
χ^2
χ^2 INV
χ^2_p
χ^2_u

SETUK
SETUSA
SIGNMT
SLOW
SSIZE4
SSIZE8
STOM
UNSIGN
WSIZE
Y.MD
🖶DLAY
🖶MODE

P.FCN
BACK
BASE?
CASE
CFALL
CLALL
CLPALL
CLREGS
CLSTK
CLα
CNST
DEC
DROP
DSL
DSZ
END
ERR
FF
FLASH?
f '(x)
f "(x)
gCLR
gDIM
gDIM?
gFLP
gPLOT
gSET
GTOα
INC
ISE
ISZ
LOADP
LOADR
LOADSS
LOADΣ
LocR
LocR?
MEM?
MSG
NOP
PopLR
PRCL
PROMPT
PSTO

STAT
COV
L.R.
SEED
SERR
$SERR_W$
SUM
s_W
s_{XY}
$\bar{x}g$
$\bar{x}_W$
$\hat{x}$
ε
ε_m
ε_P
σ
σ_W
%Σ

SUMS
nΣ
$\Sigma\ln^2 x$
$\Sigma\ln^2 y$
Σlnx
Σlnxy
Σlny
Σx
Σx^2
$\Sigma x^2 y$
Σxlny
Σxy
Σy
Σy^2
Σylnx

TEST
BC?
BS?
CNVG?
DBL?
ENTRY?
EVEN?
FC?
FC?C
FC?F
FC?S
FP?
FS?
FS?C
FS?F
FS?S
gPIX?
INTM?
INT?
KEY?
LBL?
LEAP?
M.SQR?
NaN?
ODD?
PRIME?
REALM?
SPEC?
TOP?
XTAL?
x < ?
x ≤ ?
x = +0?
x = -0?
x ≈ ?
x ≥ ?
x > ?
∞?
🖶?

PUTK	WSIZE?
RCLS	XEQα
RECV	y⇆
REGS?	z⇆
RESET	αGTO
RM?	αOFF
RTN+1	αON
R-CLR	αXEQ
R-COPY	⇆
R-SORT	🖶ADV
R-SWAP	🖶CHR
SAVE	🖶$^{C}r_{XY}$
SENDA	🖶PLOT
SENDP	🖶PROG
SENDR	🖶r
SENDΣ	🖶REGS
SKIP	🖶STK
SMODE?	🖶TAB
SSIZE?	🖶WIDTH
STOS	🖶α
TICKS	🖶α+
t⇆	🖶Σ
VIEWα	🖶+α
VWα+	🖶#

X.FCN varies with the mode set, except in programming.[64] It contains in …

… alpha mode:	… decimal mode:			… integer modes:		CPX X.FCN
VERS	$\sqrt[3]{x}$	LCM	W_m	$\sqrt[3]{x}$	RLC	$^{C}\sqrt[3]{x}$
x→α	AGM	L_n	W_p	ASR	ROUNDI	^{C}AGM
αDATE	ANGLE	LN1+x	W^{-1}	BATT	RR	^{C}CNST
αDAY	BATT	$L_n\alpha$	XNOR	CB	RRC	^{C}CONJ
αIP	B_n	LNβ	x^3	CEIL	SB	^{C}CROSS
αLENG	B_n*	LNΓ	x→α	DBLR	SEED	^{C}DOT
αMONTH	CEIL	MANT	$\sqrt[x]{y}$	DBL×	SIGN	^{C}DROP
αRC#	DATE	MAX	YEAR	DBL /	SL	$^{C}e^{x}$ -1
αRL	DATE→	MIN	αDATE	dRCL	SR	^{C}FIB
αRR	DAY	MOD	αDAY	DROP	sRCL	$^{C}g_d$
αSL	DAYS+	MONTH	αIP	FB	ULP	$^{C}g_d^{-1}$
αSR	DECOMP	NAND	αLENG	FIB	VERS	^{C}IDIV
αTIME	DEG→	NEIGHB	αMONTH	FLOOR	WHO	^{C}LN1+x
α→x	dRCL	NEXTP	αRCL	IDIV	x^3	^{C}LNβ
	DROP	NOR	αRC#	GCD	XNOR	^{C}LNΓ
	D→J	P_n	αRL	LCM	x→α	^{C}SIGN
	erf	RAD→	αRR	LJ	$\sqrt[x]{y}$	^{C}SINC
	erfc	RDP	αSL	MASKL	αIP	$^{C}W_p$
	EXPT	RESET	αSR	MASKR	αLENG	$^{C}W^{-1}$
	e^{x} -1	ROUNDI	αSTO	MAX	αRCL	$^{C}x^3$
	FIB	RSD	αTIME	MIN	αRC#	$^{C}\sqrt[x]{y}$
	FLOOR	SDL	α→x	MIRROR	αRL	^{C}β
	GCD	SDR	β	MOD	αRR	^{C}Γ
	g_d	SIGN	Γ	NAND	αSL	$^{C}(-1)^{x}$
	g_d^{-1}	SINC	γ_{XY}	nBITS	αSR	
	GRAD→	SLVQ	Γ_{XY}	NEIGHB	αSTO	
	H_n	sRCL	ΔDAYS	NEXTP	α→x	
	H_{np}	STOPW	ζ	NOR	$(-1)^{X}$	
	H.MS+	TIME	$(-1)^{X}$	RESET	×MOD	
	H.MS–	T_n	→DATE	RJ	^MOD	
	IDIV	ULP	%MG	RL		
	iRCL	U_n	%MRR			
	Iβ	VERS	%T			
	$I\Gamma_p$	WDAY	%Σ			
	$I\Gamma_q$	WHO	%+MG			
	J→D					

[64] In programming mode, these three contents are merged.

À			à		
Á			á		
ÂÃĀĂ			âãāă		
Ä			ä (ă)		
Æ			æ		
Å			å		
Ć			ć		
Č			č		
Ç			ç		
Đ			đ		
È			è		
É			é		
ÊĒĔĚ			êēĕě		
Ë			ë (ě)		
			ħ		
Ì			ì		
Í			í		
ÎĨĪĬ			îĩīĭ		
Ï			ï (ĭ)		
ÑŇ			ñň		
Ò			ò		
Ó			ó		
ÔÕŌŎ			ôõōŏ		
Ö			ö (ŏ)		
Ø			ø		
Ř			ř		
Š			š		
			ß		
Ù			ù		
Ú			ú		
ÛŨŪŬ			ûũūŭ		
Ü			ü (ŭ)		
Ů			ů		
			x̂ / x̄		
Ý			ý		
			ŷ / ȳ		
Ÿ			ÿ		
Ž			ž		

Here are the contents of the alpha catalogs. Small font is printed on light grey background on this page. The catalog CPX is listed left. Use ↑ to toggle between cases. Accented letters are as wide as plain ones wherever possible.

./,																	
,	:	;	'	"	*	@	_	~	%	$	€	£	¥	⊙	⊕	`	#

TEST									
<	≤	=	≈	≥	>	[	]	{	}

→	←	↑	↓	√	∫	∞	^	↕	

R↑																					
0	°	1	2	^{2}	^{3}	A	B	c	^{c}	d	e	k	m	n	p	q	u	w	x	^{x}	y	µ	$^{-1}$	*	∞

The letters provided in your *WP 34S* allow for correct writing the languages of more than $3 \cdot 10^9$ people using Greek or simple variants of Latin alphabets, i.e. the following languages:

Afrikaans, Català, Cebuano, Česky, Cymraeg, Dansk, Deutsch, Eesti, English, Español, Euskara, Français, Gaeilge, Galego, Ελληνικά, Hrvatski, Bahasa Indonesia, Italiano, Basa Jawa, Kiswahili, Kreyòl ayisyen, Magyar, Bahasa Melayu, Nederlands, Norsk, Português, Quechua, Shqip, Slovensky, Slovenščina, Srpski, Basa Sunda, Suomi, Svenska, Tagalog, Winaray, and Zhōngwén (with a little trick explained below). This makes the *WP 34S* the most versatile global calculator known. If you know further living languages covered, please tell us. Find the full character set provided in *Appendix E*.

Mandarin Chinese (Zhōngwén) features four tones, usually transcribed like e.g. mā, má, mǎ, and mà. So we need different letters for ā and ǎ here, and for e, i, o, and u as well. With six pixels total character height, we found no way to display these in both fonts nicely, keeping letters and accents separated for easy reading. For an unambiguous solution, we suggest using a dieresis (else not employed in Hànyǔ pīnyīn) representing the third tone here. Pinyin writers, we ask for your understanding.

Accessing Catalog Items the Fast Way

You can browse each and every catalog just using the cursors [▼] and [▲] as explained on p. 125. You may reach your target significantly faster, however, taking advantage of the alphabetical method demonstrated in the left columns of the table below:

1	User input	[CONST], [CONV], [MATRIX], [MODE], [PROB], [P.FCN], [STAT], [TEST], [SUMS], or [X.FCN]	[CPX] or [R↑] in alpha mode	[→], [TEST], or [./,] in alpha mode
	Dot matrix display	**Shows the first item in this catalog** (e.g. BC? in [TEST])	(e.g. ä in [CPX])	(e.g. ; in [./,])
2	User input	1st character of command desired (e.g. [F])	Desired basic letter (e.g. [U])	
	Dot matrix display	**Shows the first item starting with this character*** (e.g. FC?)	(e.g. Ü)	
3	User input	2nd character (e.g. [S])		
	Dot matrix display	**Shows the first item starting with this sequence*** (e.g. FS?)		
…		Continue browsing with [▼] until you reach the item desired (e.g. FS?C).	(e.g. Ü).	(e.g. €).
n	User input	[XEQ] or [ENTER↑]		
		Your *WP 34S* leaves the catalog returning to the mode set before … … and executes or inserts the command chosen, or recalls the constant selected.	… and appends the selected character to ***alpha***.	
	Dot matrix display	**Result** (e.g. true)	**Contents of alpha register** (e.g. 3 Rüben à 0,25€)	

* Like in browsing a dictionary, it may be faster to search the letter following alphabetically instead and then browse backwards. In our example, [TEST] [F][S] [▼] finds FS?C but [TEST] [G] [▲][▲][▲] would do as well.
Remember you can enter Greek letters in a search using prefix [g], e.g. [g] + [A] for α (compare p. 62).
If a character or sequence specified is not found in the catalog chosen then the first item following alphabetically will be shown – see the sorting order on p. 74. If there is no such item, then the last item in this catalog is displayed.
You may key in even more than two characters – after 3 seconds, however, or after [▼] or [▲], the search string will be reset and you may start with a first character again.

Constants (CONST)

Your *WP 34S* contains a rich catalog of constants. Navigation therein works as explained above. Names of astronomical and mathematical constants are printed on colored background below. Values of physical constants (*including their* ***relative*** *standard deviations given in parentheses below*) are from CODATA 2010, copied in July 2011, unless stated otherwise. Green background denotes exact or almost exact numeric values. The redder the background, the less precisely the particular constant is known, even by the national standards institutes and the international scientific community. [65]

For the units, remember *tesla* with $1T = 1\frac{Wb}{m^2} = 1\frac{V \cdot s}{m^2}$, *joule* with $1J = 1N \cdot m = 1\frac{kg \cdot m^2}{s^2}$

and on the other hand $1J = 1W \cdot s = 1V \cdot A \cdot s$. Thus $1\frac{J}{T} = 1A \cdot m^2$.

Employ the constants stored here for further useful equivalences, like expressing *joules* in *electron-volts* ($1A \cdot s \cdot V = \frac{1}{e}eV \approx 6.24 \cdot 10^{18} eV$), or calculating the wavelength from the frequency of electromagnetic radiation via $\lambda = c/f$, or whatever else crosses your mind.

Now here is the complete content of CONST:

	Numeric value	Remarks
1/2	0.5	Trivial but helpful constant in some iterations.
a	365.242 5 d *(per definition)*	Gregorian year
a_0	5.291 772 109 2**E**-11 m *(3.2**E**-10)*	Bohr radius $a_0 = \frac{\alpha}{4\pi \cdot R_\infty}$
a_m	384.4**E**6 m *(1**E**-3)*	Semi-major axis of the Moon's orbit
$a_\oplus$	1.495 979**E**11 m *(1**E**-6)*	Semi-major axis of the Earth's orbit. Within the uncertainty stated here, it equals 1 AU.
c	2.997 924 58**E**8 m/s *(per definition)*	Speed of light in vacuum ≈ 300 000 km/s
c_1	3.741 771 53**E**-16 $m^2 \cdot W$ *(4.4**E**-8)*	First radiation constant $c_1 = 2\pi \cdot h \cdot c^2$
c_2	0.014 387 770 $m \cdot K$ *(9.1**E**-7)*	Second radiation constant $c_2 = hc/k$

[65] The numbers printed in parentheses allow for determining the precision of results you may obtain using these constants, through the process of 'error propagation' going back to C. F. Gauß (1777 – 1855). This procedure is essential if your results are to be trustworthy – not only in science. Consult a suitable reference. Yardstick measurements cannot yield results precise to four decimals.

	Numeric value	Remarks
e	1.602 176 565**E**-19 C *(2.2**E**-8)*	Electron charge $e = \frac{2}{K_J R_K} = \Phi_0 G_0$
eE	2.718 281 828 459 045…	Euler's e. Note the letter e represents the electron charge elsewhere in this table.
F	96 485.336 5 $\frac{C}{mol}$ *(2.2**E**-8)*	Faraday's constant $F = e \cdot N_A$
Fα	2.502 907 875 095 892 8...	Feigenbaum's α and δ
Fδ	4.669 201 609 102 990 6...	
g	9.806 65 m/s^2 *(per definition)*	Standard earth acceleration
G	6.673 84**E**-11 $\frac{m^3}{kg \cdot s^2}$ *(1.2**E**-4)*	Newtonian constant of gravitation. See **GM** below for a more precise value.
G_0	7.748 091 734 6**E**-5 / Ω *(3.2**E**-10)*	Conductance quantum $G_0 = 2e^2/h = 2/R_K$
G_C	0.915 965 594 177…	Catalan's constant
g_e	-2.002 319 304 361 53 *(2.6**E**-13)*	Landé's electron g-factor
GM	3.986 004 418**E**14 m^3/s^2 *(2.0**E**-9)*	Newtonian constant of gravitation times the Earth's mass with its atmosphere included (according to *WGS84*)
h	6.626 069 57**E**-34 $J\,s$ *(4.4**E**-8)*	Planck constant
$\hbar$	1.054 571 726**E**-34 $J\,s$ *(4.4**E**-8)*	$= h/2\pi$
k	1.380 648 8**E**-23 J/K *(9.1**E**-7)*	Boltzmann constant $k = R/N_A$
K_J	4.835 978 70**E**14 Hz/V *(2.2**E**-8)*	Josephson constant $K_j = 2e/h$
l_P	1.616 199**E**-35 m *(6.0**E**-5)*	Planck length $l_P = \sqrt{\hbar G/c^3} = t_p c$
m_e	9.109 382 91**E**-31 kg *(4.4**E**-8)*	Electron mass
M_m	7.349**E**22 kg *(5**E**-4)*	Mass of the Moon
m_n	1.674 927 351**E**-27 kg *(4.4**E**-8)*	Neutron mass
m_p	1.672 621 777**E**-27 kg *(4.4**E**-8)*	Proton mass

	Numeric value	Remarks
M_P	2.176 51**E**-8 kg *(6.0**E**-5)*	Planck mass $M_P = \sqrt{\hbar c / G} \approx 22\mu g$
m_u	1.660 538 921**E**-27 kg *(4.4**E**-8)*	Atomic mass unit $= 10^{-3}\ kg / N_A$
$m_u c^2$	1.492 417 954**E**-10 J *(4.4**E**-8)*	Energy equivalent of atomic mass unit
m_μ	1.883 531 475**E**-28 kg *(5.1**E**-8)*	Muon mass
$M_\odot$	1.989 1**E**30 kg *(5**E**-5)*	Mass of the Sun
$M_\oplus$	5.973 6**E**24 kg *(5**E**-5)*	Mass of the Earth
N_A	6.022 141 29**E**23 $/ mol$ *(4.4**E**-8)*	Avogadro's number
NaN	not numeric	'Not a Number', e.g. $\ln(x)$ for $x \le 0$ or $\tan(90°)$ unless in complex domain. So NaN covers poles as well as regions where a function result is not defined at all. Note that infinities, however, are considered numeric (see the end of this table). Non-numeric results will lead to an error message thrown – unless flag **D** is set.
p_0	101 325 Pa *(per definition)*	Standard atmospheric pressure
q_P	1,875 545 9**E**-18 $A\ s$ *(6.0**E**-5)*	Planck charge $q_p = \sqrt{4\pi\varepsilon_0 \hbar c} \approx 11.7e$. This was in CODATA 2006, but in 2010 no more.
R	8.314 462 1 $\frac{J}{mol \cdot K}$ *(9.1**E**-7)*	Molar gas constant
r_e	2.817 940 326 7**E**-15 m *(9.7**E**-10)*	Classical electron radius $r_e = \alpha^2 \cdot a_0$
R_K	25 812.807 443 4 Ω *(3.2**E**-10)*	von Klitzing constant $R_K = h / e^2$
R_m	1.737 530**E**6 m *(5**E**-7)*	Mean radius of the Moon
R_∞	1.097 373 156 853 9**E**7 $/ m$ *(5.0**E**-12)*	Rydberg constant $R_\infty = \alpha^2 m_e c / 2h$
$R_\odot$	6.96**E**8 m *(5**E**-3)*	Mean radius of the sun
$R_\oplus$	6.371 010**E**6 m *(5**E**-7)*	Mean radius of the Earth
Sa	6.378 137 0**E**6 m *(per definition)*	Semi-major axis of the model *WGS84* used to define the Earth's surface for GPS and other surveying purposes
Sb	6.356 752 314 2**E**6 m *(1.6**E**-11)*	Semi-minor axis of *WGS84*

	Numeric value	Remarks
Se^2	6.694 379 990 14**E**-3 *(1.5**E**-12)*	First eccentricity squared of *WGS84*
Se'^2	6.739 496 742 28**E**-3 *(1.5**E**-12)*	Second eccentricity squared of *WGS84* (it is really called e'^2 in that article, sorry)
Sf^{-1}	298.257 223 563 *(per definition)*	Flattening parameter of *WGS84*
T_0	273.15 K *(per definition)*	= 0°C, standard temperature
t_P	5.391 06**E**-44 s *(6.0**E**-5)*	Planck time $t_P = \sqrt{\frac{\hbar G}{c^5}} = \frac{l_p}{c}$
T_P	1.416 833**E**32 K *(6.0**E**-5)*	Planck temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_p c^2}{k} = \frac{E_p}{k}$
V_m	0.022 413 968 m^3/mol *(9.1**E**-7)*	Molar volume of an ideal gas at standard conditions $V_m = RT_0/p_0 \approx 22.4\frac{l}{mol}$
Z_0	376.730 313 461… Ω	Characteristic impedance of vacuum $Z_0 = \mu_0 c$
α	7.297 352 569 8**E**-3 *(3.2**E**-10)*	Fine-structure constant $\alpha = \frac{e^2}{4\pi\varepsilon_0\hbar c} \approx \frac{1}{137}$
γEM	0.577 215 664 901 532 86…	Euler-Mascheroni constant γ_{EM}
γ_p	2.675 222 005**E**8 $\frac{1}{s \cdot T}$ *(2.4**E**-8)*	Proton gyromagnetic ratio $\gamma_p = 2\mu_P/\hbar$
ε_0	8.854 187 817…**E**-12 $\frac{A\,s}{V\,m}$	Electric constant or vacuum permittivity $\varepsilon_0 = \frac{1}{\mu_0 c^2}$
λ_C	2.426 310 238 9**E**-12 m *(6.5**E**-10)*	Compton wavelengths of the electron $\lambda_C = \frac{h}{m_e c}$, neutron, and proton, respectively
λ_{Cn}	1.319 590 906 8**E**-15 m *(8.2**E**-10)*	
λ_{Cp}	1.321 409 856 23**E**-15 m *(7.1**E**-10)*	
μ_0	1.256 637 061 4…**E**-6 $\frac{V\,s}{A\,m}$	Magnetic constant or vacuum permeability $\mu_0 := 4\pi \cdot 10^{-7}\frac{V \cdot s}{A \cdot m}$
μ_B	9.274 009 68**E**-24 J/T *(2.2**E**-8)*	Bohr's magneton $\mu_B = e\hbar/2m_e$
μ_e	-9.284 764 30**E**-24 J/T *(2.2**E**-8)*	Electron magnetic moment

	Numeric value	Remarks
μ_n	-9.662 364 7**E**-27 J/T *(2.4**E**-7)*	Neutron magnetic moment
μ_p	1.410 606 743**E**-26 J/T *(2.4**E**-8)*	Proton magnetic moment
μ_u	5.050 783 53**E**-27 J/T *(2.2**E**-8)*	Nuclear magneton $\mu_u = e\hbar/2m_p$
μ_μ	-4.490 448 07**E**-26 J/T *(3.4**E**-8)*	Muon magnetic moment
σ_B	5.670 373**E**-8 $\frac{W}{m^2K^4}$ *(3.6**E**-6)*	Stefan-Boltzmann constant $\sigma_B = \frac{2\pi^5k^4}{15h^3c^2}$
Φ	1.618 033 988 749 894…	Golden ratio $\Phi = \frac{1+\sqrt{5}}{2}$
Φ_0	2.067 833 758**E**-15 $V\,s$ *(2.2**E**-8)*	Magnetic flux quantum $\Phi_0 = h/2e = 1/K_J$
ω	7.292 115**E**-5 rad/s *(2**E**-8)*	Angular velocity of the Earth according to *WGS84*
$-\infty$	-InFinity	May the Lord of Mathematics forgive us calling these 'constants'. Note both are counted as numeric values in your *WP 34S*.
∞	InFinity	
#		See the very last command in the *IOP*.

Unit Conversions (CONV)

CONV mainly provides the means to convert local to common units.[66] Navigation works as in the other catalogs. There is one specialty, however: [f] [B] (i.e. [1/x]) will execute the inverse of the conversion displayed and leave CONV.

Example: Assume the display set to FIX 3. Then keying in

[4] [h] [CONV] [A] will display

telling you that 4 *acres* equal 1.619 *hectares*.

Now press [f] [B] and you will get

being the amount of *acres* equaling 4 *hectares*.

Press [h] [CONV] again and you see

confirming what was just said.

Leave CONV via [EXIT] and the display will return to 9.884.

The conversions provided are listed in the table below. The calculations are for your orientation only; your *WP 34S* uses more precise numbers where applicable.

Conversion	Calculation	Remarks	Class
°C→°F	* 1.8 + 32		Temperature
°F→°C	- 32) / 1.8		Temperature
°→G	/ 0.9	Converts to *gon* , also known as *grads* or *gradians*	Angle
°→rad	* π / 180	Equals D→R, converting to *radians*	Angle
acres→ha	* 0.404 686	1 $ha = 10^4 m^2$	Area
acreUS→ha	* 0.404 687	Based on the U.S. Survey foot	Area
ar.→dB	$20\lg(a_1/a_2)$	Amplitude ratio	Ratio
atm→Pa	* 1.013 25E5		Pressure

[66] The *SI* system of units is agreed on internationally. Meanwhile, it is adopted by all countries on this planet except two. Thus for most readers, most of the units appearing in CONV may look obsolete at least. They die hard, however, in some corners of this world (English is spoken in all of them). For symmetry reasons, we think about adding some traditional Indian and Chinese units to CONV.
This table may also give you an idea of the mess we had in the world of measures before going metric.

Conversion	Calculation	Remarks	Class
AU→km	* 1.495 98E8	Astronomic units	Length
bar→Pa	* 1E5		Pressure
Btu→J	* 1 055.06	British thermal units	Energy
cal→J	* 4.186 8		Energy
cft→l	* 28.316 9	Cubic feet	Volume
cm→inches	/ 2.54		Length
cwt→kg	* 50.802 4	(Long) hundredweight = 112 lbs	Mass
dB→ar.	$10^{R_{dB}/20}$	Amplitude ratio	Ratio
dB→pr.	$10^{R_{dB}/10}$	Power ratio	Ratio
fathom→m	* 1.828 8		Length
feetUS→m	* 0.304 801	1 U.S. Survey foot = $^{1200}/_{3937}\,m$ exactly	Length
feet→m	* 0.304 8	The so-called 'international feet'	Length
flozUK→ml	* 28.413 1	1 l = $^{1}/_{1000}\,m^3$	Volume
flozUS→ml	* 29.573 5		
galUK→l	* 4.546 09		
galUS→l	* 3.785 42		
G→°	* 0.9		Angle
g→oz	/ 28.349 5		Mass
G→rad	* π / 200		Angle
g→tr.oz	/ 31.103 5		Mass
ha→acres	/ 0.404 686	1 ha = 10000 m^2	Area
ha→acreUS	/ 0.404 687	Based on the U.S. Survey foot	Area
HP_e→W	* 746	Electric horsepower	Power
hpUK→W	* 745.700	British horsepower	Power
inches→cm	* 2.54		Length
inHg→Pa	* 3 386.39		Pressure

Conversion	Calculation	Remarks	Class
J→Btu	/ 1 055.06		Energy
J→cal	/ 4.186 8		Energy
J→kWh	/ 3.6**E**6		Energy
kg→cwt	/ 50.802 4	(Long) hundredweight = 112 *lbs*	Mass
kg→lb	/ 0.453 592		Mass
kg→stones	/ 6.350 29		Mass
kg→s.cwt	/ 45.359 2	Short hundredweight = 100 *lbs*	Mass
km→AU	/ 1.495 98**E**8	Astronomical units	Length
km→l.y.	/ 9.460 73**E**12	Light years	Length
km→miles	/ 1.609 344		Length
km→nmi	/ 1.852	Nautical miles	Length
km→pc	/ 3.085 68**E**16	Parsec	Length
kWh→J	* 3.6**E**6		Energy
lbf→N	* 4.448 22		Force
lb→kg	* 0.453 592		Mass
l.y.→km	* 9.460 73**E**12	Light years	Length
l→cft	/ 28.316 9	$1\ l = {}^{1}/_{1000}\ m^3$	Volume
l→galUK	/ 4.546 09		
l→galUS	/ 3.785 42		
miles→km	* 1.609 344		Length
ml→flozUK	/ 28.413 1	$1\ ml = 1\ cm^3$	Volume
ml→flozUS	/ 29.573 5		
mmHg→Pa	* 133.322		Pressure
m→fathom	/ 1.828 8		Length
m→feet	/ 0.304 8		Length
m→feetUS	/ 0.304 801		Length
m→yards	/ 0.914 4		Length

Conversion	Calculation	Remarks	Class
nmi→km	* 1.852	Nautical miles	Length
N→lbf	/ 4.448 22		Force
oz→g	* 28.349 5	Ounces	Mass
Pa→atm	/ 1.013 25**E**5	1 Pa = 1 N/m^2	Pressure
Pa→bar	/ 1**E**5		Pressure
Pa→inHg	/ 3 386.39		Pressure
Pa→mmHg	/ 133.322		Pressure
Pa→psi	/ 6 894.76		Pressure
Pa→torr	/ 133.322		Pressure
pc→km	* 3.085 68**E**16	Parsec	Length
pr.→dB	$10\lg\left(P_1/P_2\right)$	Power ratio	Ratio
psi→Pa	* 6 894.76	Pounds per square inch	Pressure
PS(hp)→W	* 735.499	Horsepower	Power
rad→°	* 180 / π	Equals R→D	Angle
rad→G	* 200 / π		Angle
stones→kg	* 6.350 29		Mass
s.cwt→kg	* 45.359 2	Short hundredweight = 100 lbs	Mass
s.tons→t	* 0.907 185	Short tons	Mass
tons→t	* 1.016 05	Imperial tons	Mass
torr→Pa	* 133.322	1 $torr$ = 1 $mm\ Hg$	Pressure
tr.oz→g	* 31.103 5	Troy ounces	Mass
t→s.tons	/ 0.907 185	1 t = 1000 kg	Mass
t→tons	/ 1.016 05		

Conversion	Calculation	Remarks	Class
W→HPe	/ 746		Power
W→hpUK	/ 745.700		Power
W→PS(hp)	* 735.499		Power
yards→m	* 0.914 4		Length

The constant $\mathbf{T_o}$ may be useful for conversions of temperatures, too; it is found in *CONST* and is not repeated here because it is only added or subtracted.

You may, of course, combine conversions as you like. For **example**, filling your tires with a maximum pressure of 30*psi* the following will help you at gas stations in Europe and beyond:

[3] [0] [h] [CONV] [P] [S] [XEQ]
[h] [CONV] [P] [▼] [XEQ] resulting in 2,1*bar*.

Now you can set the filler and will not blow your tires.

In cases of emergency of a particular kind, remember *becquerel* equals *hertz*, *gray* is the unit for deposited or absorbed energy ($1Gy = 1J/kg$), and *sievert* (*Sv*) is *gray* times a radiation dependant dose conversion factor for the damage caused in human bodies.

In this area also some outdated units may be found in older literature: Pour les amis de Mme. *Curie*, $1Ci = 3.7 \cdot 10^{10} Bq = 3.7 \cdot 10^{10} decays/s$. And for those admiring the very first Nobel laureate in physics, Mr. *Röntgen*, for finding the x-rays (ruining his hands in these experiments), the charge generated by radiation in matter was measured by the unit $1R = 2.58 \cdot 10^{-4} As/kg$. A few decades ago, the *rem* (i.e. *roentgen equivalent in man*) measured what the *sievert* does today.

Predefined Global Alpha Labels (CAT)

In addition to the label 'δx' reserved for step size in calculation of derivatives (see f'(x) in the *IOP*), additional labels may already be provided for particular tasks. You will find them listed in CAT when the respective library routines are loaded in *FM* (see below). Thus they will not take any steps from user program memory in *RAM*.

All library routines presently available are found on the *WP 34S* website in the directory *http://wp34s.svn.sourceforge.net/viewvc/wp34s/library/* . They are text files with extension `.wp34s` by convention. These routines include a suite of basic 3D vector operations, a TVM (time value of money) application, some matrix routines including an editor, and more. You may open these text files using e.g. Notepad, and you should find the necessary user information at the beginning of each file.

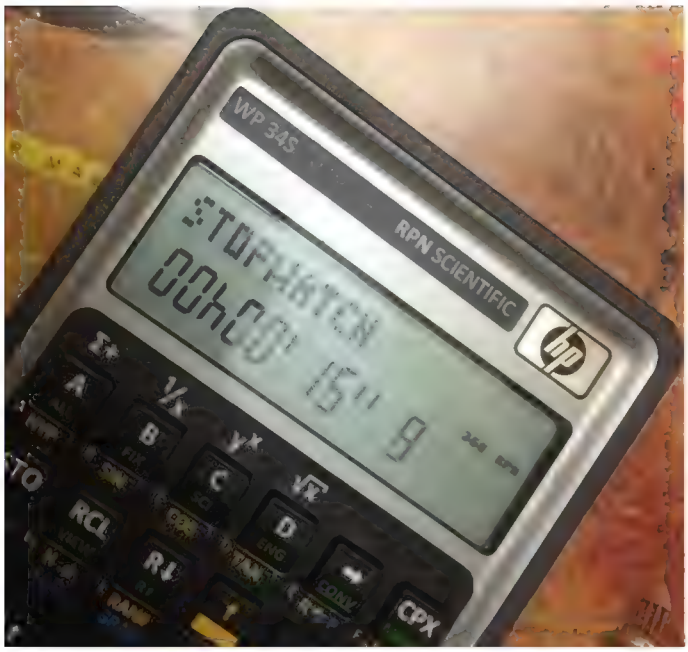
The library files are also included in the distribution *ZIP* file in source form (`*.wp34s`) as well as in a precompiled library (`wp34s-lib.dat` – some 3kB) which is part of the firmware files `calc_full.bin`, `calc_xtal_full.bin`, and `calc_ir_full.bin` – so you get the full library when you load one of these firmware files into your *WP 34S* (see *Appendix A* about how to do this).

When you copy `wp34s-lib.dat` into the directory your *WP 34S* emulator runs in, you can access all those routines via CAT from your emulator as well. See *Appendix D*.

See *Appendix H* if you want to write such routines yourself.

THE STOPWATCH APPLICATION

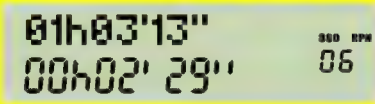
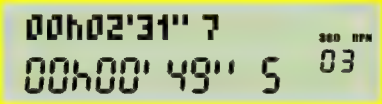
As mentioned in the *IOP*, a stopwatch is provided. It works on your *WP 34S* with a quartz crystal and associated firmware installed (see *App. A* and *H*). It will work on the emulator as well.



Name	Keys to press	in modes	Remarks
STOPW	h X.FCN STOPW CPX R/S	DECM, ¬STO	Stopwatch following the *HP-55* timer.[67] When STOPW is started, the display will look like this: STOPWATCH 00h00' 00'' 0 unless the timer was already running.
	Within STOPW, the following keys will work:		
	R/S		starts or stops the timer without changing its value.
	CLx or ⇦		resets the timer to zero without changing its status (running or stopped).
	EEX		hides or displays tenths of seconds. Startup default is 'display'.
	n n		sets the *current register address* (*CRA*, startup default is 00). Your numeric input will already be displayed in the exponent section as shown here[68]: STOPWATCH 27h31' 55'' 5 01

[67] There are two deviations: Your *WP 34S* will not take the content of **X** as start time. Start times are supported by RCL here. And your *WP 34S* will display tenths instead of hundredth of seconds.

[68] Attempts to specify a *CRA* beyond the allocated address range will be blocked and may cause '- -' or the like being displayed in the exponent section.
On the *HP-55*, input of a single digit was sufficient for storing, since only 10 registers were featured for this purpose there. Furthermore, there was no automatic address increment.

[ENTER↑]	stores the present timer value in the current register at execution time in format `hhh.mmssd` without changing the timer status or value. It then increments the *CRA* and displays it as shown above.
[➡]	hides or displays the *CRA*. Startup default is 'hide'.
[▲] or [▼]	increments or decrements the *CRA*, respectively.
[.]	combines [ENTER↑] and [CLx] in one keystroke, but the total time since the last explicit press of [CLx] or [⬅] is shown and updated in the top row like:  or  . Note this total time is volatile, however – it will disappear without a trace when [CLx] or [⬅] is pressed.
[A]	adds the present timer value to the statistics registers like [Σ+] would do. This allows for computing its arithmetic mean and standard deviation after leaving STOPW.
[+]	combines [A] and [.] in one keystroke.
[RCL] ***nn***	recalls ***rnn*** without changing the status of the timer. The value recalled may be used e.g. as start time for further incrementing.
[EXIT]	leaves the application. Unless already stopped, however, the timer continues incrementing in the background (indicated by the small '=' annunciator flashing) until a) stopped explicitly by [R/S] within STOPW or b) your *WP 34S* is turned off. **ATTENTION**: Since your *WP 34S* shuts down automatically after some five minutes of non-usage, press some prefix keys twice in suitable intervals if you need to measure longer times. While the stopwatch display is limited to 99h59' 59'' 9, internal counting will continue with the display showing the time modulo 100.
Note <u>no other keys</u> will work in STOPW – so e.g. for adding or subtracting split times you have to leave this application.	

Note STOPW is not programmable.

APPENDIX A: SETUP AND COMMUNICATION

How to Flash Your *HP-20b* or *HP-30b*

Unless you buy a *WP 34S* pre-flashed as explained on p. 5, you must do the flashing yourself. Then you need an unmodified *HP-20b* or *HP-30b* calculator, a special cable, a binary file to load on your computer, and software for the transmission to your calculator.

- The transmission software is called *MySamBa*. Download `MySamBa.zip` from *http://sourceforge.net/projects/wp34s/files/FlashTool/* and unpack it.

- The specific binary file you need to transmit to your calculator to make it your *WP 34S* is called `calc.bin` and is included in the zipped release package you can download from *http://sourceforge.net/projects/wp34s/files/* . Alternatively, you may download from *http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/realbuild/* one of the following files alone:

`calc.bin`	features maximum *FM*, but supports neither the real time clock commands (DATE, TIME, SETDAT, and SETTIM) nor STOPW nor the print commands,
`calc_full.bin`	equals `calc.bin` but already includes the FM library;
`calc_xtal.bin`	assumes a quartz crystal is installed and includes the real time clock commands and STOPW but no print commands,
`calc_xtal_full.bin`	equals `calc_xtal.bin` but includes the FM library;
`calc_ir.bin`	contains everything featured in `calc_xtal.bin` plus the print commands requiring both the quartz crystal and an *IR* diode built in,
`calc_ir_full.bin`	equals `calc_ir.bin` but includes the FM library.

  The amount of *FM* you have to pay for the STOPW application is 1.5kB, for printer support some 3.5kB more. Make your choice! But check the hardware pre-requisites (see *Appendix H*) or your calculator may hang.

- For the cable, there are two alternatives, presented in order of appearance:

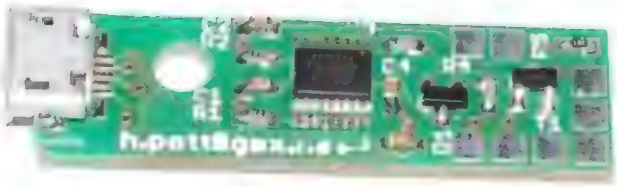
  A. There was a limited production run of special programming cables supplied by *HP*. This item – pictured overleaf – is almost out of stock now. Thus if you have got one, keep it. Then you will need a computer with a traditional 9-pin serial port.

     ATTENTION: If your computer does not feature such an hardware serial interface (many modern computers do not), you will need a *USB*-to-serial converter to connect this programming cable to your computer. Following our

experience, converters containing *FTDI* chips work – others may not. Such a converter is offered in the web e.g. at *http://commerce.hpcalc.org/usbserial.php* .

WARNING: As long as this cable is connected to your calculator, it may draw a considerable current from its batteries. If your calculator happens to hang during the flashing procedure described overleaf, the calculator processor may be left running at full speed, draining your coin cells while you are trying to find out what is going wrong. Thus, disconnect your cable when you will not need it for the next few minutes. For frequent flashing, an external $3V$ *DC* power supply may quickly pay for itself. Take care to connect '+' to the outer and '–' to the inner battery contact. The flashing will work best with a stable $3V$ supply – but as long as the voltage supplied exceeds $2.5V$ it is ok.

B. You may use an ordinary micro *USB* cable for the connection to your *PC* if you install a small custom board developed by *Harald Pott*, hosting a *USB* port. With the board pictured here, your calculator will be powered through that *USB* line while connected.



Thus you will not need the special programming cable nor an hardware serial port, and avoid the battery drain observed – but you will need to install this board. See *Appendix H* for more information and a detailed installation guide.

Having prepared your calculator, computer, software, file, and cable, follow one of the two alternative procedures A or B below for transforming an *HP-20b* or *HP-30b* into a *WP 34S*.

GENERAL WARNING: Flashing your *HP-20b* or *-30b* will erase the *HP* firmware – your business calculator will then be gone. Its firmware will be replaced by the file you downloaded. Thereafter you will have a *WP 34S RPN Scientific* – i.e. your calculator will react as documented in this manual.[69]

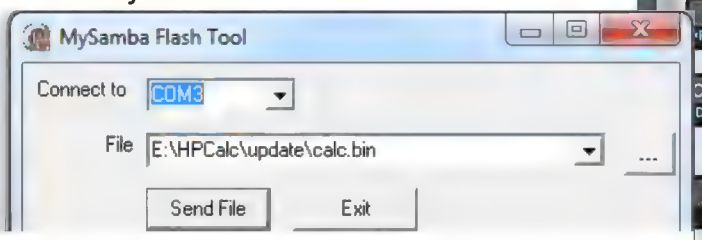
This also means your calculator will not do anything useful for you after step 6 and before step 9 is successfully completed in the procedures described below. Your calculator may even look dead – it is not, be assured. If the procedure is interrupted at any time, don't worry: remember RESET will always turn your calculator off, and [ON] shall turn it on[70] – simply start over at step 1.

[69] If – for any reason whatsoever – you want to return to the original *HP* business calculator firmware, you can do so by sending the respective binary file instead of `calc.bin` or its siblings. An *HP-30b* file is hosted on our website for example – note it represents the firmware status as known to us today and will not be maintained.

[70] … as long as the batteries are good – even if you do not see anything on the *LCD*.

A1. Remove the black battery door of your calculator. Connect the programming cable to the programming port of the calculator and to your computer.

A2. Start *MySamBa*. Pick the port you use for your cable and select the file you want to transmit. The window may look like:



Do **not** start sending yet! Just leave the window open.

A3. Press [ON/CE] to turn the calculator on.

A4. Hold down the ERASE button on the cable (do not release it until step A7).

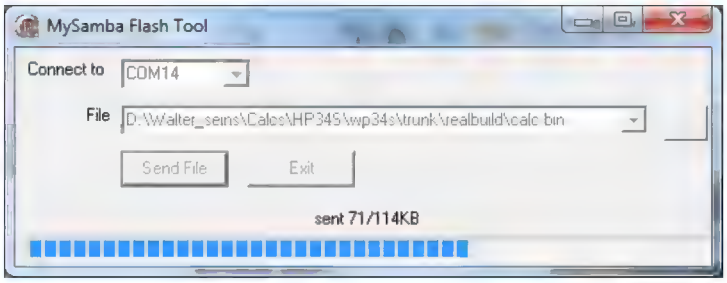
A5. Press the RESET button on the cable. It will turn the calculator off.

A6. Press [ON/CE] to turn the calculator on again.

A7. Release ERASE now. Press RESET to turn the calculator off again.

A8. Press [ON/CE] to turn the calculator on again. It will look dead. Do not worry – see the note above.

A9. Now click "Send File" in the *MySamBa* window and wait for it to finish transmission (it will take some 20*s*). If you have the *FTDI USB/serial adapter* mentioned above, you will see the blue TX light blinking. If *MySamBa* should fail[71], return to A3.



A10. Press RESET to turn the calculator off once more. Exit *MySamBa.* Disconnect the cable. Close the battery door.

A11. Press and release [ON/CE] . Your calculator should turn on as a *WP 34S* now.

[71] Then it throws (a) "Error, could not connect to calculator" or (b) "Unable to connect". Typical causes are: (a) a wrong COM port or not the proper driver for that port, (b) the cable makes no proper contact or your calculator is not turned on. Anyway, this dead looking calculator is a most annoying state since you do not get the slightest feedback from your system while you keep looping through the steps above. And with the programming cable connected, your battery will be drained in this time … so press RESET, disconnect, think, and check before reconnecting and retrying. If it fails repeatedly, turn to p. 182.

If you have installed the custom *USB* board mentioned above before transforming your *HP-20b* or *-30b* into a *WP 34S*, your calculator will be powered through this port while connected to your computer. The standard *USB* cables, however, do not feature the buttons ERASE and RESET of the custom programming cable. Thus, the flashing procedure will work here as follows:

B1. Remove the battery door. Connect said micro *USB* cable to the *USB* port of the calculator and to your computer.

B2. Start *MySamBa*. Enter the proper port and file information as explained in step A2 above (see also p. 197). Do not start sending yet – just leave the window open.

B3. Press [ON/CE] to turn the calculator on.

B4. Shorten the top right and the bottom left of the six pads below the RESET hole by a removable link (a paper clip works well).

B5. Use a suitable pin to press the RESET button behind the hole. It will turn your calculator off.

B6. Ensure the link makes good contact to both pads. Then press [ON/CE] to turn the calculator on again. It will erase the old firmware.

B7. Remove the shorting link. Press RESET to turn your calculator off again.

B8. Press [ON/CE] on the calculator. It will look dead like on the picture. Do not worry – see the note above.

B9. Click "Send File" in the *MySamBa* window and wait for it to finish transmission (it will take some 20*s*). If it fails[71], return to B3.

B10. Press RESET to turn the calculator off once more. Exit *MySamBa.* Close the battery door.

B11. Press and release [ON/CE] . Your calculator should turn on as a *WP 34S* now. You can disconnect the cable unless you want to use it for data transfer from your emulator (see *Appendix D*).

Overlays – Where to Get Them and How to Make Them Yourself if Required

After flashing successfully, a keyboard overlay is very helpful for further work since most labels deviate from those on said business calculators.

You get fine adhesive vinyl overlays from *Eric Rechlin* at http://commerce.hpcalc.org/overlay.php.

You can choose if you want them with white labels on the prefix keys (the original design, shown left) or with f, g, and h in their respective colours (shown above). Supporters of the 'Great American Divide' can apply them with a ÷ instead of a / as well.

Eric's vinyls come pre-stamped. Detach the key plate part of it from the carrier paper first. Adjust it carefully over your *WP 34S* and attach it starting with the section between ENTER and STO, then proceeding towards the top and bottom. The vinyl shall not contact the keys. After covering the key plate, detach the individual key labels and apply them to the respective keys. If you need to lift a label once attached – e.g. for readjusting it – a sharp knife will serve you best: put it between key and label and lift a corner of the label, then take it with your finger nails or tweezers and peel it off.

If *Eric's* vinyls are not available for any reason, you can make preliminary paper overlays most easily using the picture shown left. Print this page to scale (the picture shall be 68*mm* wide), cut it out, span it over your *WP 34S* using transparent adhesive tape, and you are done – the flexibility of this setup allows actuating the keys though there are no holes in your overlay. It will serve you well until the vinyls will arrive.

Updating Your *WP 34S*

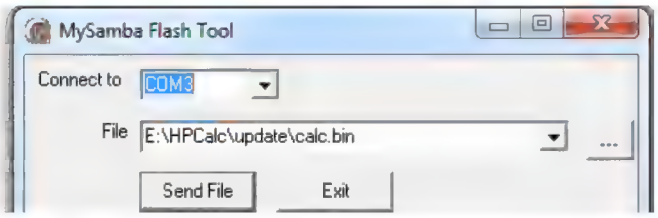
Whether you bought your *WP 34S* pre-flashed or did the first flashing yourself, you may want to keep it up-to-date when a new release is published or after you modified its hardware. **We recommend you SAVE your work each time before updating** (your *WP 34S* will show `Saved` when successful). Starting the new firmware, this backup will be restored automatically – if you didn't accidently press the ERASE button on the programming cable.

With the proper binary downloaded as described on p. 147, updating works as follows:

1. Remove the battery door of your *WP 34S*.
2. Connect the programming cable to the programming port of your *WP 34S* and to your computer. **Do not touch the cable buttons in the remaining procedure!**

2. Connect the micro *USB* cable to the *USB* port of your *WP 34S* and to your computer.

3. Start *MySamBa*. Enter the proper port and file information (see p. 147ff).

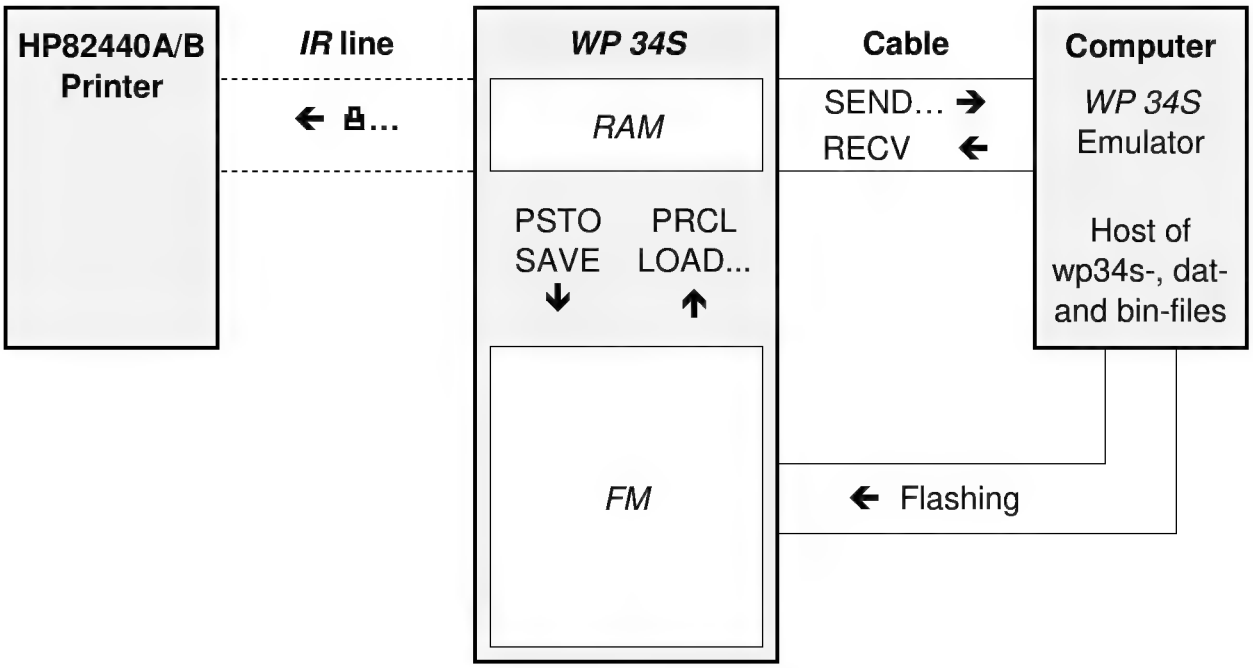
   

   Do **not** start sending yet! Just leave the window open.
4. Press [ON] to turn your *WP 34S* on. Then press [ON] + [D] simultaneously. It is echoed `Debug ON`. Release both keys.[72]
5. Now press [ON] + [S] simultaneously. It is echoed `SAM-BA? boot`. Release only [S] and press it a second time for confirmation, then release both keys. This resets the Samba boot bit and turns your *WP 34S* off.
6. Press [ON] . Your *WP 34S* will look dead. Do not worry – it is not.
7. Click "Send File" in the *MySamBa* window and wait for it to finish transmission (it will take some 20*s*).[73]
8. Use a suitable pin to press the RESET button behind the hole on the back of your *WP 34S*. Your device will turn off. Exit *MySamBa*. Disconnect the cable. Close the battery door.
9. Press [ON] . Your *WP 34S* should turn on now with the last backup `Restored` and the new firmware loaded. Check using VERS.

[72] The next step will only work in debug mode. See *Appendix H*.
[73] If *MySamBa* fails for any reason, press RESET and disconnect the programming cable. Then take your time and check footnote 71. Return to step 6 when you want to retry.

I/O Overview

Looking to I/O from your *WP 34S calculator*, this matter now appears as follows schematically:



SAVE is for battery-fail-safe internal backup of your work as explained on p. 72. The different flavors of LOAD are for recovering from backup. Alternatively to the commands SAVE and LOAD, you may hold down [ON] (i.e. [EXIT]) and press one of the following keys twice:

[STO] for backup:	Creates a copy of the *RAM* in *FM* like SAVE does.
[RCL] for restore:	Restores the most recent backup like LOAD does.

PRCL recalls an individual program from the *FM* library into *RAM*. PSTO stores a copy of an individual program from *RAM* into the *FM* library. Note that by recalling a library routine from *FM* via PRCL, editing it, and storing it back via PSTO you may modify this part of the *FM* library. See the *IOP* for these commands and *Appendix B* for details about the *RAM* and *FM* sections of your *WP 34S*.

The different flavors of SEND are for sending your work from *WP 34S RAM* to your computer, where you will enjoy more editing comfort. See *Appendix D* for more.

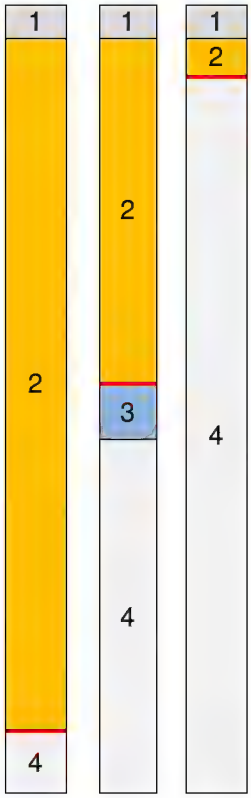
APPENDIX B: MEMORY MANAGEMENT

Your *WP 34S* features 6*kB* of *RAM* (of which 4*kB* are volatile) and 128*kB* of *FM*. The firmware takes some 90% of *FM*, depending on the file you loaded in flashing. The remaining, user accessible part of *FM* may be almost 14*kB*. Of these, 2*kB* are reserved for the backup region corresponding to the 2*kB* of non-volatile *RAM* featured; the remainder may be filled by a collection (a.k.a. *library*) of different programs.

The remaining part of this appendix covers *RAM* only. It discusses how the available memory is divided in program area, local and global data. The 1024 *words* of non-volatile *RAM* are shared by four sectors:

1. Status and configuration data
2. Global registers, i.e. general purpose registers and stack
3. Registers used for cumulative statistics (optional)
4. Program memory including subroutine return stack (*SRS*, including local registers if applicable).

These sectors are arranged top down as pictured at right to scale. The center bar shows the *RAM* configuration in startup default (but with statistics registers allocated). The left bar shows it for maximum register space allowed (but without statistics), the right for maximum program space. This appendix covers the variable boundaries between these sectors.[74]



Status and Configuration Data

This sector of 88 *bytes* is fixed at the top of available memory and is completely user transparent. It contains status and modes data, the 30 *byte* alpha register, and 14 *bytes* holding the 112 global user flags.

Global Registers

Global registers are placed near the end of available memory. In startup default memory layout, the numbered registers **R00** to **R99** precede the twelve stack and special registers **X, Y, Z, T, A, B, C, D, L, I, J,** and **K** as shown on p. 24. This totals to 112 global registers, which is the maximum available. Their number can be reduced down to the 12 lettered registers using REGS (see the *IOP*). REGS? will return an integer between 0 and 100 corresponding to the number of global numbered registers currently allocated.

REGS controls the lower boundary of the global register sector (abbreviated *LBG* in the following). Reducing the number of registers will pull up *LBG* to higher absolute addresses;

[74] A complete copy of the *RAM* as you configured and loaded it can be written to *FM* using SAVE. It will remain safely preserved not only when your *WP 34S* turns off but also even when its batteries fail. See *Appendix A* for more information about flashing and handling of *FM* in general.

increasing their number will push it down. The memory contents are moved accordingly, thus preserving the data in the surviving registers. Contents of deallocated registers are lost; newly added registers are cleared. The lettered registers do not move.

Example: See the global register sector at startup default in the following memory table. The two rightmost columns show what happens after subsequent execution of REGS 96 and REGS 98. The registers are loaded with arbitrary values here so they can be traced easily. *LBG* is indicated by a red horizontal line.

	Default startup memory allocation		After executing REGS 96		Then after executing REGS 98	
Absolute address	Contents	Relative register address	Contents	Relative register address	Contents	Relative register address
X+11	*k = 40.7*	**R111 = K**	*40.7*	**K = R111**	*40.7*	**K = R111**
…	**…**	**…**	**…**	**…**	**…**	**…**
X+2	*z = 4.5*	**R102 = Z**	*4.5*	**Z = R102**	*4.5*	**Z = R102**
X+1	*y = -33.8*	**R101 = Y**	*-33.8*	**Y = R101**	*-33.8*	**Y = R101**
X	*x = 123.0*	**R100 = X**	*123.0*	**X = R100**	*123.0*	**X = R100**
X-1	*r99 = -13.6*	**R99**	*23.1*	**R95**	*0.0*	**R97**
X-2	*r98 = 67.9*	**R98**	*6.4*	**R94**	*0.0*	**R96**
X-3	*r97 = -45.2*	**R97**	**4.8**	**R93**	*23.1*	**R95**
X-4	*r96 = 9.7*	**R96**	**…**	**…**	*6.4*	**R94**
X-5	*r95 = 23.1*	**R95**	**…**	**…**	*4.8*	**R93**
X-6	*r94 = 6.4*	**R94**	**…**	**…**	**…**	**…**
X-7	*r93 = 4.8*	**R93**	**…**	**…**	**…**	**…**
…	**…**	**…**	**…**	**…**	**…**	**…**
X-94	*r06 = 62.4*	**R06**	*5.7*	**R02**	*29.4*	**R04**
X-95	*r05 = -0.6*	**R05**	*-2.4*	**R01**	*81.3*	**R03**
X-96	*r04 = 29.4*	**R04**	*1.1*	**R00**	*5.7*	**R02**
X-97	*r03 = 81.3*	**R03**			*-2.4*	**R01**
X-98	*r02 = 5.7*	**R02**			*1.1*	**R00**
X-99	*r01 = -2.4*	**R01**				
X-100	*r00 = 1.1*	**R00**				
…						

Note the absolute addresses of **R00** up to **R*n-1*** change after REGS ***n*** whenever ***n*** is changed, while their contents are copied. The lettered registers stay at fixed absolute addresses.

In indirect addressing, zero in the index register points to **R00** always. Index values exceeding the maximum set by REGS will throw an 'out of range' error, unless they fall between 100 and 111 – where the lettered registers live.

The two sectors following in lower memory (summation registers and *SRS*) are tied to *LBG* – their contents will be copied whenever *LBG* moves. This makes it possible to execute REGS in the middle of a subroutine without disrupting the program.

Summation Registers

The memory needed for cumulative statistics is allocated separately – these data are no longer held in global general purpose registers. This allows for higher internal precision and prevents destroying these data inadvertently. The only way to update statistical data is via Σ+ and Σ- . The accumulated data are evaluated and recalled by dedicated commands; they are not accessible by STO or RCL.

The first invocation of Σ+ allocates 70 *words* for the 14 summation registers.[75] They are inserted between *LBG* and *SRS*, pushing the latter down in memory. Depending on the competing requirements for program and data space, it may be necessary to make room first (see overleaf).

After CLΣ, CLALL, or RESET, the memory allocated for the summation registers is released. All pointers are automatically adjusted, so the memory allocation or release will not disrupt a running program. Recall commands such as e.g. Σxy or SUM will return zero if no data are allocated; other statistical operations will throw an error if not enough data are present.

> **ATTENTION:** The summation data will be cleared automatically when a long program is loaded (from *FM* or via the serial interface) if the registers would no longer fit in *RAM* after that load. You can avoid this by reducing the amount of numbered registers using REGS before the load attempt. This should move the summation data out of the way.

Subroutine Return Stack (*SRS*) and Program Memory

Both share the remaining space at lowest memory addresses.

The ***SRS*** is used for return addresses and local data. Its upper boundary is given by *LBG* or the lowest summation register if applicable. There is no command to set the size of the *SRS* – it fills all the space down to the top program step currently stored. When new program steps are entered, the *SRS* is reset, not only to make room but because any stored address may become invalid by changing the program.

Local data are pushed on the *SRS*. Thus they cannot overwrite global data; this greatly increases the flexibility of programs. LOCR ***n*** allocates ***n*** local registers and a fixed amount of 16 local flags. It does so by pushing a frame on the *SRS* containing a marker, a flag word, and the registers requested (0 to 144). The marker contains the frame size in *words*, depending on the precision mode set (see *Appendix H*). A pointer to this frame in memory is initialized. If the pointer is zero, no local registers exist. Newly allocated registers are cleared.

[75] Herein, 2 words are employed for Σn, 4 × 8 words for Σx^2, Σy^2, Σxy, and Σx^2y, and 9 × 4 words for the other sums. If memory allocation for these 70 words fails, an error will be thrown.

Calling LOCR again in the same subroutine will adjust the number of local registers. This requires data copying since these registers are allocated from low to high addresses and the *SRS* grows in the opposite direction. LOCR? will return the number of local registers currently allocated in the routine you are in.

See *overleaf* for addressing local data, and for an example of recursive programming. The *SRS* must be large enough to hold these data, however, so you may have to make room first – see next paragraph.

Below the *SRS*, **program memory** holds the stored program steps. A typical program step takes just one word. Multi-byte labels and multi-character alpha strings take two *words* each. The total size of program memory depends on the number of global and local registers allocated, as explained in the following.

Making Room for Your Needs

The 12 special (lettered) registers are always allocated. The *SRS* has a minimum size of six *words* or levels. Everything else is user distributable within the 982 *words* left for sections 2 to 4, so:

982 = ***r*** + ***s*** + ***p*** with

- ***r*** = number of *words* allocated for global registers. These are 4 per standard register. There are at least 12 and at most 112 of them. So ***r*** varies between 48 and 896 (this maximum is explained *in Appendix H*); startup default is 448.
- ***s*** = number of *words* allocated for summation registers (70 if they are used; startup default is 0).
- ***p*** = number of *words* available for program steps and *SRS*. One step is already taken by the inevitable final END statement; 6 *words* is the minimum size of the *SRS*. So STATUS will show you a maximum of 933 free *words* in *RAM*, meaning up to 927 free program steps. Startup default is 533 steps. Subroutine nesting and local registers expand the *SRS*, thus reducing the program space available.

If, for instance, you need to do statistics and also use 20 global numbered registers, there will be space for 777 program steps maximum.

You have several options for increasing the free space where you need it (see the picture on p. *154*):

1. Reduce the number of global numbered registers allocated. One register less typically allows for four additional program steps.
2. Move programs to *FM* and clear the respective steps in *RAM*. Four cleared program steps typically allow for one additional register.
3. Release the summation registers when you do not need them anymore. This space may be distributed to up to 70 additional program steps, up to 17 additional registers, or a mix.

Which solution serves you best depends on your application. You may of course combine options. Use STATUS to monitor the free space available and the amount of global and local numbered registers allocated.

Addressing and Accessing Local Data, Recursive Programming

Global data take relative addresses from 0 to 111 as described on p. 154. So, relative addresses of local data begin with 112 and may go up to 255 if 144 local registers are allocated. The first 16 local registers and all local flags may also be directly addressed using a dot heading the number – the arguments go from .00 to .15, corresponding to relative addresses from 112 to 127.[76] Any registers beyond are only indirectly addressable. This scheme allows for indirectly addressing

- a global register via a global index register (e.g. STO→23 with ***r23*** < 112),
- a global register via a local index register (e.g. STO→.15 with ***r.15*** < 112),
- a local register via a global index register (e.g. STO→47 with ***r47*** ≥ 112), and
- a local register via a local index register (e.g. STO→.06 with ***r.06*** ≥ 112).

Subroutine calls: XEQ – executed in a program – just pushes the return address on the *SRS* before it branches to the target. The subroutine called will keep having access to the caller's local data as long as it does not execute LOCR itself. As soon as it does, the pointer to the local data is newly set, and the subroutine called cannot access the caller's local data anymore.

RTN or POPLR – executed in a program – check if the current *SRS* pointer points to a local frame (as explained on p. 156). If true then the pointer is moved above that frame, and the *SRS* is searched from this point upwards for another local frame. If such a frame is found then its pointer is stored; otherwise the pointer to the active local frame is cleared. RTN will branch to the return address found, while POPLR will just continue execution. So the current local frame is dropped and the next higher (or older) frame is reactivated if one exists.

Manually executing RTN, starting a new program with XEQ, SLV, etc., or program editing will clear the *SRS* and remove all local registers and flags by clearing the pointer. All such data are lost then!

Recursive programming: Using local registers allows for creating a subroutine that calls itself recursively. Each invocation deals with its local data only. Of course the *RPN* stack is global so be careful not to corrupt it.

> Here is a recursive implementation of the factorial. It is an **example** for demonstration only, since this routine will neither set the stack correctly nor will it work for input greater than some hundred:

[76] Only arguments up to 127 are storable in an op-code, hence the limit.

```
LBL 'FAC'
IP
x≤1?
GTO 00
1
RTN
LBL 00
LocR 001
STO .00
DEC X
XEQ 'FAC'
RCL× .00
RTN
```

Assume $x = 4$ when you call FAC. Then it will allocate 1 local register (**R.00**) and store 4 therein. After decrementing x, FAC will call itself.

Then FAC_2 will allocate 1 local register (**R.00$_2$**) and store 3 therein. After decrementing x, FAC will call itself again.

Then FAC_3 will allocate 1 local register (**R.00$_3$**) and store 2 therein. After decrementing x, FAC will call itself once more.

Then FAC_4 will return to FAC_3 with $x = 1$. This x will be multiplied by ***r.00$_3$*** there, returning to FAC_2 with $x = 2$. This x will be multiplied by ***r.00$_2$*** there, returning to FAC with $x = 6$, where it will be multiplied by ***r.00*** and will finally become 24.

Switching between Standard Real (*SP*) and Integer Modes

Your *WP 34S* starts in standard real mode (DECM) when you get it new. You may use it for integer computations as well, as shown above many times. Going from DECM to any integer mode, the values on the current stack will be truncated to integers. Going from integer mode to DECM, the current stack contents (being all integers) will be converted to decimal. All other memory contents will stay as they were!

See the fate of some register contents undergoing mode switches in the following **examples**, where ***j***, ***k***, ***r00***, and ***r01*** <u>will be checked by recalling</u> them. :

	X	Y	J	K	R00	R01
Contents at start e.g.	1.1	20.2	300.3	4000.4	50000.5	600000.6
After 2COMP, WSIZE 32, BASE 10	1 d	20 d	3075 d	40964 d	512005 d	6291462 d
Recall the registers by sRCL			300 d	4000 d	50000 d	600000 d
567 STO J, -9 STO 00	-9 d	567 d	567 d	40964 d	-9 d	6291462 d
DECM	-9.0	567.0	5.7 $^{-396}$	4000.4	3.0 $^{-389}$	600000.6
Recall the registers by iRCL			567.0	40964.0	-9.0	6291462.0

Note that identical register contents are interpreted quite differently in DECM and integer modes. Even very small integers may lead to very large surprises:

	R00	R01	R02
Contents at start e.g.	0	2	10
Then after WSIZE 64, BASE 16	0 h	2238000000000002 h	223C00000000000 1 h
Recall the registers by sRCL	0 h	2 h	A h
2 STO 01, A STO 02	0 h	2 h	A h
DECM	0	2 $^{-398}$	1 $^{-397}$
Recall the registers by iRCL	0	2	10

Thus take care with indirect addressing!

Example: Start with DECM, WSIZE 64, 2COMP, 0 STO 00, 2 STO 01, 10 STO 02 as above. So RCL→01 shall recall ***r02***. Let us check. Key in:

RCL → 0 1	10.	just for verification: OK
g 16	A h	turn to BASE 16
RCL → 0 1	Out of range Error	compare the table above
2 STO 0 1	2 h	
RCL → 0 1	000000000001 h	now this works in hex mode, too
f ◀	223C h	
f H.d	246684669589 18	turn to DECM
RCL → 0 1	0.	compare the table above – indirection has to ignore all fractional components. Otherwise ISG/DSE loops all simply would not work.

All this is caused by the **internal representation of numbers:** While *integers* are simply stored as such (allowing for $n \leq 1.84 \cdot 10^{19}$ in UNSIGN or $|n| \leq 9.22 \cdot 10^{18}$ in the other modes using 64 bits), *standard floating point numbers* are stored using a format as follows:

- Real zero is stored as integer zero, i.e. all bits cleared.
- The mantissa of a real number (also known as *significand* in this context) is encoded in five groups of three digits. Each such group is packed into 10 bits straight forward, meaning e.g. $555_{10} = 10\ 0010\ 1100_2$ or $999_{10} = 11\ 1110\ 0111_2 = 3E7_{16}$. So the 15 rightmost decimal digits of the *significand* take the least significant 50 bits. Trailing zeroes are omitted, so the *significand* will be right adjusted.
- The most significant (64th) bit takes the sign of the mantissa.
- The remaining 13 bits are used for the exponent and the leftmost digit of the mantissa. Of those 13, the lowest 8 are reserved for the exponent. For the top 5 bits it gets complicated[77]: if they read …
  - `00ttt`, `01ttt`, or `10ttt` then `ttt` takes the leftmost digit of the *significand* ($0 - 7_{10}$), and the top two bits will be the most significant bits of the exponent;
  - `11uut` then `t` will be added to 1000_2 and the result (8_{10} or 9_{10}) will become the leftmost digit of the *significand*. If `uu` reads 00, 01, or 10 then these two will be the most significant bits of the exponent. If `uu` reads 11 instead, there are codes left for encoding special numbers (e.g. infinities).

  In total, we get 16 digits for the mantissa and a bit less than 10 bits for the exponent: its maximum is $10\ 1111\ 1111_2$ (i.e. 767_{10}). For reasons becoming obvious below, 398 must be subtracted from the value in this field to get the true

[77] Don't blame us – this part follows the standard IEEE 754.

exponent of the number represented. The 16 digits of the *significand* allow for a range from 1 to almost 10^{16}.

Rewarding your patience so far, we will show you some illustrative examples of the encoding in your *WP 34S* instead of telling you more theory:

Floating point number	Hexadecimal value stored	Bottom bits split in groups of 10	Top 14 bits in binary representation	Stored exponent
1.	22 38 00 00 00 00 00 01		0010 0010 0011 10	398
-1.	A2 38 00 00 00 00 00 01		1010 0010 0011 10	398
111.	22 38 00 00 00 00 00 6F		0010 0010 0011 10	398
111.111	22 2C 00 00 00 01 bC 6F	06F 06F	0010 0010 0010 11	395
-123.000123	A2 20 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0010 00	392
$9.99 \cdot 10^{99}$	23 bC 00 00 00 00 03 E7		0010 0011 1011 11	495
$1 \cdot 10^{-99}$	20 AC 00 00 00 00 00 01		0010 0000 1010 11	299
$1 \cdot 10^{-383}$	00 3C 00 00 00 00 00 01		0000 0000 0011 11	15

The last number is the smallest that can be entered directly. Dividing it by 10^{15} results in $1 \cdot 10^{-398}$, being stored as hexadecimal 1. Divide this by 1.999 999 999 99 and the result will remain $1 \cdot 10^{-398}$ in default rounding mode (and in RM 1, 2, 3, and 5, see RM). Divide it by 2 instead and the result will become zero.

Look at the other end:

9.999 999 999 999 $99 \cdot 10^{384}$	77 FF E7 F9 FE 7F 78 00	9 3E7 3E7 3E7 3dE 000	0111 0111 1111 11	767

This number (featuring the digit 9 twelve times) is the greatest that can be entered numerically. Adding $9.999 \cdot 10^{372}$ to it will display $1 \cdot 10^{385}$ …

$1 \cdot 10^{385}$	77 FF E7 F9 FE 7F 9F E7	9 3E7 3E7 3E7 3E7 3E7	0111 0111 1111 11	767

… which is stored as 9.999 999 999 999 $999 \cdot 10^{384}$. This is the greatest number representable. All this follows *Decimal64* floating point format, though not exactly.

Additionally, your *WP 34S* features three special numbers:

Infinity	78 00 00 00 00 00 00 00		0111 1000 0000 00	n/a
-Infinity	F8 00 00 00 00 00 00 00		1111 1000 0000 00	n/a
not numeric	7C 00 00 00 00 00 00 00		0111 1100 0000 00	n/a

These special numbers are legal results on your *WP 34S* if flag **D** is set.

APPENDIX C: MESSAGES AND ERROR CODES

There are some commands generating messages, be they in the numeric or in the dot matrix section of the display. Of these, DAY, DAYS+, ERR, STATUS, VERS, and WDAY were introduced above in the *section about display*. Others are PROMPT, αVIEW and more alpha commands, and the test commands (see p. *67*). Also two *constants* will return a special message when called.

Furthermore, there are a number of error messages. Depending on error conditions, the following messages will be displayed in the mode(s) listed:

Message	Error code	Mode(s)	Explanation and examples
Bad time or dAtE	2	DECM	Invalid date format or incorrect date or time in input, e.g. month >12, day >31 .
Bad digit Error	9	Integer	Invalid digit in integer input, e.g. 2 in binary or 9 in octal mode. Will be displayed as long as the respective key is pressed.
Bad mode Error	13	All	Caused by calling an operation in a mode where it is not defined, e.g. calling a constant in a program written in DECM but executed in an integer mode.
Domain Error	1		An argument exceeds the domain of the mathematical function called. May be caused by roots of negative numbers or logs of $x \le 0$ (both if not preceded by CPX), by $0/0$, $\Gamma(0)$, $\tan(90°)$ and equivalents, by $\operatorname{artanh}(x)$ for $\lvert\mathrm{Re}(x)\rvert \ge 1$, by $\operatorname{arcosh}(x)$ for $\mathrm{Re}(x) < 1$, etc.[78]
Flash is Full	23	All	No more space in *FM*. Delete a program from *FM* to regain space.
Illegal OPErAtion	7	All	May appear in an attempt running an old program containing a command which turned nonprogrammable after said program was written.
Invalid dAtA	18	All	Set when there is a checksum error either in *FM* or as part of a serial download. It is also set if a *FM* segment is otherwise not usable.
Invalid PArAMEtEr	16		Similar to error 1 but a parameter specified in **J** or **K** is out of valid range for the function called. May appear e.g. if LgNrm is called with $j < 0$.

[78] Note that e.g. tan(90°) and logs of 0 are legal if flag D is set. See the end of this appendix.

Message	Error code	Mode(s)	Explanation and examples
I/O Error	17	[illegible]	See *Appendix D*.
Matrix MISMATCH	21	DECM	• A matrix isn't square when it should be. • Matrix sizes aren't miscible.
No crystal INSTALLED	24	All	May be thrown by DATE, TIME, SETDAT, or SETTIM. See *Appendix H* for curing.
No root Found	20	DECM	The solver did not converge.
No such LABEL	6	All	Attempt to address an undefined label.
Out of range Error	8	All	• A number exceeds the valid range. This can be caused by specifying decimals >11, word size >64, negative flag numbers, integers $\geq 2^{64}$, *hours* or *degrees* >9000, invalid times, denominators ≥9999, etc. • A register or flag address exceeds the valid range of currently allocated registers. May also happen in indirect addressing or calling nonexistent local addresses. • An R-operation (e.g. R-COPY) attempts accessing invalid register addresses. • A matrix *descriptor* would go beyond the registers available or a row or column index is too large.
RAM is Full	11	All	No more space in *RAM*. May be caused by attempts to write too large programs, allocate too many registers, and the like. May happen also in program execution due to too many local data dynamically allocated (see p. *156*).
Singular Error	22	DECM	• Attempt to use a LU decomposed matrix for solving a system of equations. • Attempt to invert a matrix when it isn't of full rank.
Stack CLASH	12	All	STOS or RCLS attempt using registers that would overlap the stack. Will happen with e.g. SSIZE = 8 and STOS 94 (if REGS 100 is set).

Message	Error code	Mode(s)	Explanation and examples
Too few dAtA PoInts	15	DECM	A statistical calculation was started based on too few data points, e.g. regression or standard deviation for < 2 points.
Too long Error	10	All	Keyboard input is too long for the buffer. Will happen e.g. if you try to enter more than 12 digits.
Undefined OP-CodE	3	All	An instruction with an undefined operation code occurred. Should never happen – but who knows?
Word size too SMALL	14	Integer, ¬STO	Register content is too big for the word size set.
Write ProtEctEd	19	All	Attempt to delete or edit program lines in *FM*.
+∞ Error	4	¬α, ¬STO, ¬flag D	• Division of a number > 0 (or < 0) by zero. • Divergent sum or product or integral. • Positive (or negative) overflow in DECM (see p. 40). • Error 5 is also thrown for a logarithm of (+)0, while a logarithm of -0 returns NaN.
−∞ Error	5		
∫ ⁜ 1,30	25	DECM & RCL	This is not an error but only a message. It will be returned by MSG 25 and works exclusively in programs.

Each error message is *temporary* (see p. 38), so ⬅ or EXIT will erase it and allow continuation. Any other key pressed will erase it as well, but will also execute with the stack contents present. Thus, another easy and safe return to the display shown before the error occurred is pressing an arbitrary prefix twice.

A final note about flag **D**: if it is set, errors 4 and 5 will not occur at all, and error 1 will happen less frequently, since ±∞ and NaN are legal results then (see the respective entry in *CONST* and the very end of *Appendix B*).

APPENDIX D: THE *WP 34S* EMULATOR ON YOUR COMPUTER

For the emulator, run `\trunk\windows\bin\wp34sgui.exe` – it features the identical function set as your *WP 34S* calculator. While tactile feedback, boot speed, pocketability and battery life suffer, some things become easier, e.g. printing (see below). What else is different?

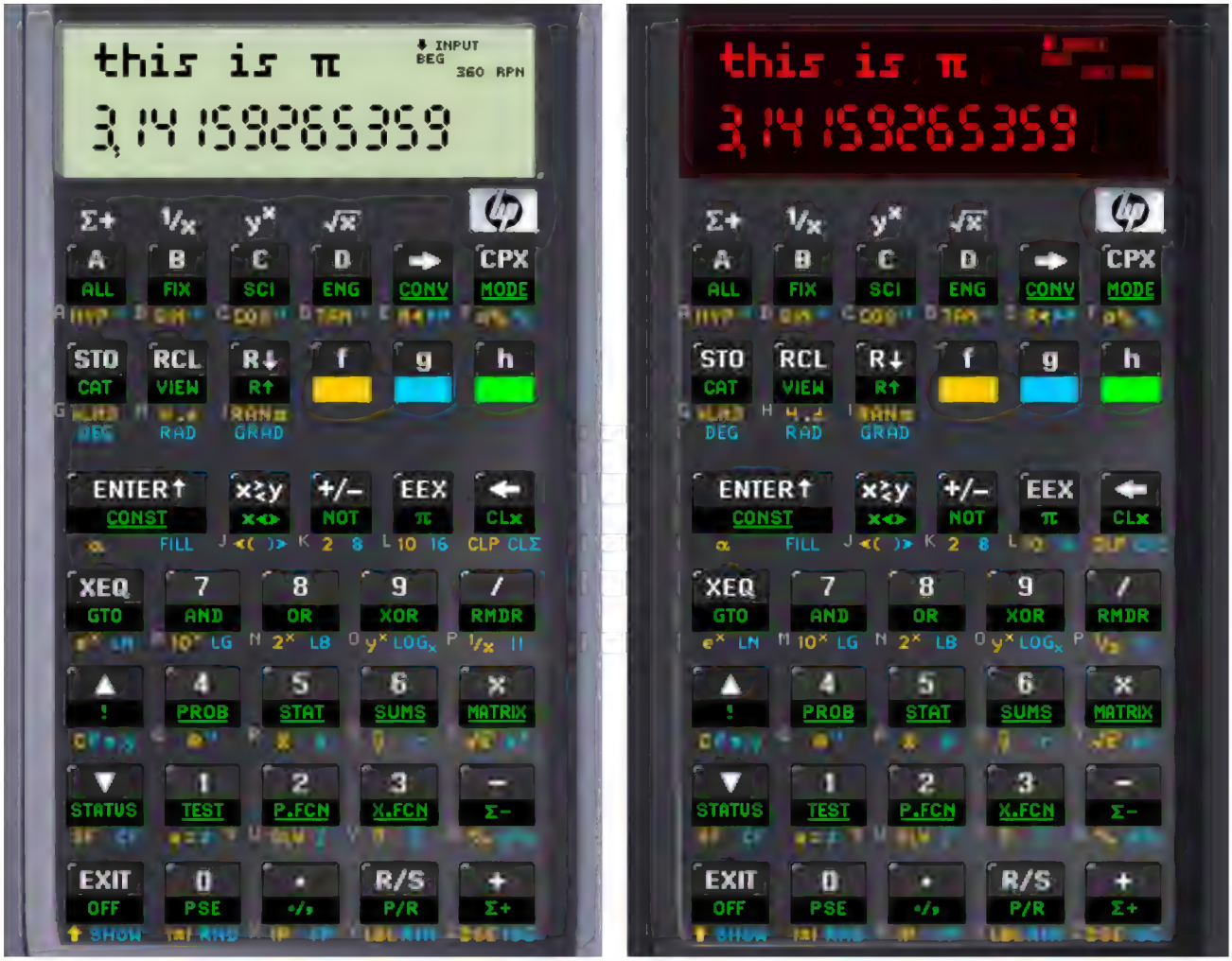
Typically the emulator is operated by the computer mouse. You click on the respective areas of the keyboard image instead of pressing keys. Right clicking on a key area is a shortcut to its green h-shifted label. And there is one important extra: a right click on the WP logo top right opens a menu as seen here.

It contains links to the project website and the manual (under <Help>) as well as to three different skins for the emulator.

The 'medium' skin is shown several times in this manual; the other two are smaller and are displayed overleaf. All three skins are printed to scale here.

Furthermore, this menu allows you to import or export numbers, export ***alpha*** as text or the complete '*LCD*' screen as a picture to the clipboard for use in other computer applications.

If your computer has a numeric keypad, the digit and arithmetic operation keys there are shortcuts to the corresponding 'keys' of the emulator. The up and down cursor keys are connected to [▲] and [▼], **Backspace** and **Del** to [←], and **Enter** and **CR** to [ENTER↑].



Data Transfer between Your *WP 34S* and Your Computer

The emulator and the calculator can talk to each other over the cable used for flashing. Either one of the cables mentioned on p. 147 will do, or you need a modified calculator as described here:

http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv020.cgi?read=186826 .

Remember the programming cable draws current from the batteries of your *WP 34S*, so disconnect it from your *WP 34S* as soon as it is no longer needed.

In the emulator directory, a plain ASCII text file `wp34s.ini` must be placed that contains just a single line (terminated by CR/LF) stating the name of the computer port used for serial communication, such as e.g.

COM2:

The following commands (all stored in P.FCN catalog) allow for sending programs, registers or all *RAM* from your *WP 34S* to your computer or vice versa. With a suitable cable it would be even possible to transfer data directly between two *WP 34S.*

On the receiving device, enter RECV. It will display Wait... .

On the sender you have three alternatives:

1. SENDP will send the current program from *RAM*. It will display Wait... as well. After successful transmission, the sender will show OK and the receiver will display Program .
2. SENDR will send all the global numbered registers allocated, displaying Wait... . After successful transmission, the sender will show OK and the receiver will display Register .
3. SENDA will send the complete two *kilobytes* of non-volatile *RAM*, displaying Wait... . After successful transmission, the sender will show OK and the receiver will display All RAM .

   If your *WP 34S* receives data from the emulator on your computer, these data will be stored in the respective *RAM* section directly. If the emulator receives data from your *WP 34S*, these data may be used immediately as well, and will be part of the file `wp34s.dat` which is written when exiting the emulator (see overleaf).

The little = annunciator is lit while the serial port is in use. Take [EXIT] to abort the communication if necessary.

Each of the four commands RECV, SENDP, SENDR, and SENDA features a fixed timeout of some 10*s* for setting up the connection. After an interval of inactivity of said length, an I/O Error is thrown indicating no communication has occurred. If such an error appears in the middle of a transmission, try again.

Using a *WP 34S* <u>without</u> the crystal installed, you may also get an I/O error because of its baud rate setting being a bit too far off. To determine the speed, use the loop

```
CLx
INC X
BACK 001
```

and let it run for 30*s* while connected to the computer. The expected result at nominal speed is around 123,000 for v3.1. On a crystal-less *WP 34S*, the I/O commands accept a correction factor in percent in **X**. Divide the counts you measured by 123,000 and leave it in **X** for RECV or SEND… Already some 2% may make a significant difference. Values between 80 and 120 are accepted – all other are ignored. On the emulator or on a *WP 34S* with the crystal installed and initialized, ***x*** will be ignored in those commands.

With your data transmitted successfully from your *WP 34S* to your computer, see section 4 of the separate *WP 34s Assembler Tool Suite User Guide* for the way to extract `.wp34s`-files from the file `wp34s.dat`. The `.wp34s`-files may be opened and edited by a suitable text file editor, e.g. WordPad. After completion of editing and assembling the routine(s) – and after successful tests on the emulator – you may

revert the transmission: issue RECV on your *WP 34S* and send the respective modified data from your computer back to your *WP 34S.* What works on the emulator shall work on your calculator as well (provided the necessary hardware is installed) – both feature the same set of functions. Your *WP 34S* is one of the smallest calculators in a commercial package allowing such a bidirectional data exchange and backup.

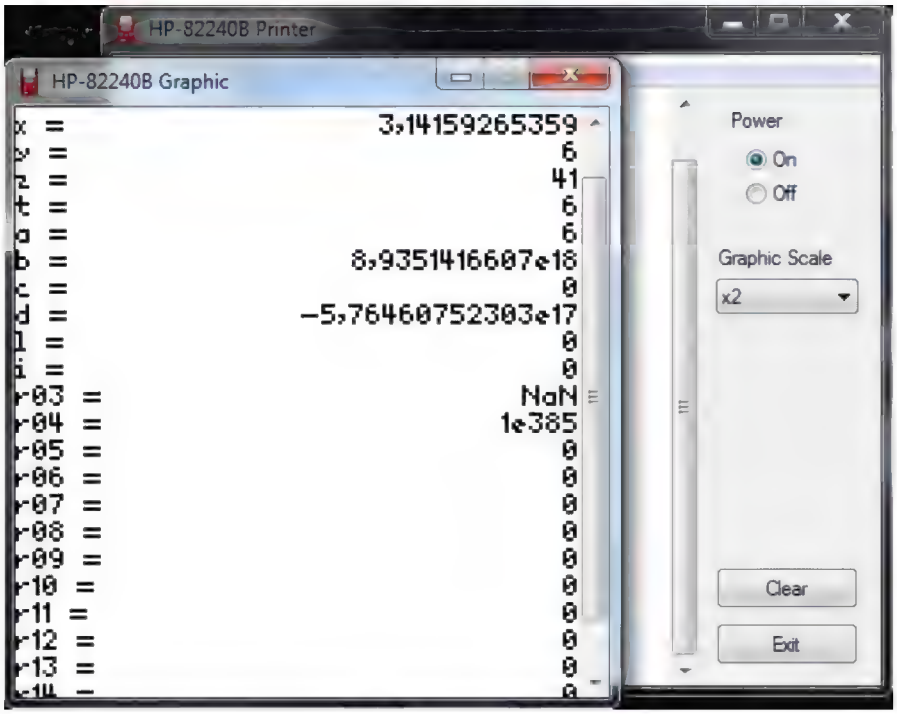
The newer Qt-based emulators for Windows and MacOS contain a setup option for the serial interface. They will eventually replace the current Windows emulator. with the commands above.

Mapping of Memory Regions to Emulator State Files

Region		State file	Remarks
Backup	buP	`wp34s-backup.dat`	This file is created by SAVE.
Flash library	Lib	`wp34s-lib.dat`	This file is written whenever a flash command is executed.
RAM	rAM	`wp34s.dat`	Backup of the emulator *RAM* area (registers, state, and programs) – this file is written only when exiting the emulator.

All these files are only read into memory at emulator startup.

Simulating a Printer on Your Computer



For simulated printing, use *Christoph Gießelink's HP82240B Printer Simulator*, available here: http://hp.giesselink.com/hp82240b.htm .

Installation and use are self-explanatory. It is an autonomous application, so you have to start and exit it independently. The 'print' shown here was generated by PSTK and PREGS. Guess what stack size was set?

APPENDIX E: CHARACTER SETS

The following table shows the complete dot matrix character set as implemented in big and small font sorted according to the hexadecimal character codes (Unicode). Characters with codes $<20_{16}$ are for control purposes – some of them (4, 10_{10}, 27_{10}) may be useful for *HP82240B* control.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
002x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
008x																																
009x																																
00Ax				£		¥	|			¨			¬							£		¥	|			¨			¬			
00Bx	¤	±	²	³		µ	µ										¤	±	²	³		µ	µ									
00Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
00Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00Fx	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý		ÿ	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý		ÿ
010x	Ā	ā	Ă	ă			Ć	ć					Ĉ	ĉ			Ā	ā	Ă	ă			Ć	ć					Ĉ	ĉ		
011x			Ē	ē	Ĕ	ĕ					Ē	ē							Ē	ē	Ĕ	ĕ					Ē	ē				
012x								ħ	Ī	ī	Ī	ī	Ī	ī										ħ	Ī	ī	Ī	ī	Ī	ī		
013x																																
014x								Ñ	ñ				Ō	ō	Ō	ō								Ñ	ñ				Ō	ō	Ō	ō
015x									Ŕ	Ŕ															Ŕ	Ŕ						
016x	Ŝ	Ŝ							Ū	Ū	Ū	Ū	Ū	Ū	Ů	Ů	Ŝ	Ŝ							Ū	Ū	Ū	Ū	Ū	Ū	Ů	Ů
017x								Ŷ	Ŷ					Ž	Ž									Ŷ	Ŷ					Ž	Ž	
023x				Ȳ																Ȳ												
039x		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
03Ax	Π	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω								Π	Ρ	Σ	Τ	Υ	Φ	Χ	Ψ	Ω							
30Bx		α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο		α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
03Cx	π	ρ	σ	τ	υ	φ	χ	ψ	ω								π	ρ	σ	τ	υ	φ	χ	ψ	ω							
1D6x		ˣ			ᵤ			ᵧ										ˣ			ᵤ			ᵧ								
1E8x	x̄											x̄					x̄											x̄					
201x									'	'	'		"	"	"										'	'	'		"	"	"	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
207x																																
208x																																
209x																																
20Ax																																
219x																																
21Cx																																
221x																																
222x																																
223x																																
224x																																
225x																																
226x																																
239x																																
249x																																
24Ax																																
24Bx																																
24Cx																																
24Dx																																
260x																																
264x																																

In addition, there are two raster fonts supplied. Characters printed on darker grey show identical patterns in all four fonts. Characters from 00F0 to 00FF (printed on yellow here) cannot be the last ones in a group of three (see LBL etc.). Note the internal character sorting in your *WP 34S* differs from Unicode – the characters of these four dot-matrix fonts are repeated in the internal order at codes greater than E000. Do <u>not</u> use these high codes in texts, however, since some programs (e.g. Preview) may not read them correctly.

For the numeric seven-segment display and the annunciators top right, there is another character set and font provided:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Remarks
002x																	
003x																	
004x																	Note M
005x																	Note W
006x																	Note m
007x																	Note w
008x																	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Remarks
009x																	
00Ax																	For STATUS
00Bx																	For high bits in BASE 2
00Cx																	
00Dx																	
00Ex	↓	INPUT	=	▬	BEG	STO	RCL	RAD	360	RPN							Annunciators
00Fx																	Segments

All fonts are available at
http://wp34s.svn.sourceforge.net/viewvc/wp34s/trunk/windows/winfont/ .

APPENDIX F: CORRESPONDING OPERATIONS TO THE *HP-42S* AND *-16C* FUNCTION SETS

In the *IOP*, the corresponding functions of vintage *HP* calculators were mentioned under the respective entry of your *WP 34S*. The tables below revert this in a way: the first shows the functions of the *HP-42S* and the corresponding ones of your *WP 34S* unless they carry identical names and are either both keyboard accessible or both stored in a catalog or menu. There is an analog table for the *HP-16C* functions on p. *178* below. **Functional differences of homonymous commands are covered in the *IOP*.**

HP-42S

Remarks printed on light grey indicate commands being either default settings or keyboard accessible on your *WP 34S* while you must use a menu on the *HP-42S*.

HP-42S	***WP 34S***	**Remarks**
ABS		Press [|x|].
ACOSH		Press [HYP⁻¹][COS].
ADV	🖶ADV	
AGRAPH	n/a	The *LCD* of the HP-30b has only a very small dot matrix section (see p. *35*). Striving for full fledged graphics therein would be futile.
AIP	αIP	
ALENG	αLENG	
ALL		Press [ALL].
ALLΣ	Superfluous	Your WP 34S always runs in ALLΣ mode.
[f] [ALPHA]	[α]	See the description of alpha mode on p. *61*.
AND		Press [AND].
AOFF	αOFF	
AON	αON	
ARCL	αRCL or αRC#	
AROT	αRL or αRR	
ASHF	αSL or αSR	
ASINH		Press [HYP⁻¹][SIN].
[f] [ASSIGN]	n/a	The *LCD* of the *HP-30b* does not allow soft keys. This prevents us from offering you a CUSTOM menu.
ASTO	αSTO	Press [f][STO] in alpha mode.
ATANH		Press [HYP⁻¹][TAN].
ATOX	α→x	
AVIEW	αVIEW	

HP-42S	*WP 34S*	**Remarks**
[orange] BASE	Superfluous	HEXM, DECM, OCTM, BINM, as well as the LOGIC operations AND, OR, XOR, and NOT are on the keyboard of your *WP 34S* in all modes but alpha. A…F are provided in *integer modes* if applicable. The functions corresponding to BIT? and ROTXY are then in X.FCN.
BASE+ etc.	Superfluous	Your *WP 34S* executes these arithmetic commands in integer modes automatically.
BEEP	n/a	The *HP-30b* does not contain the necessary hardware.
BINM	BASE 2	Press [2].
BIT?	BS?	
[orange] CATALOG	Replaced	This is split in *various catalogs* on your *WP 34S.*
CF		Press [CF].
CLA	CLα	Press [CLx] in alpha mode.
CLD	Superfluous	Any keystroke will clear a *temporary message*.
[orange] CLEAR	[P.FCN]	CLΣ, CLP, CLST, CLA, and CLX are on the keyboard.
CLKEYS	n/a	See ASSIGN.
CLLCD	n/a	Use [α] [CLx] to clear the dot matrix section. See AGRAPH.
CLMENU	n/a	See ASSIGN.
CLP		Press [CLP].
CLRG	CLREGS	
CLST	CLSTK	Press [0] [FILL].
CLV	n/a	The hardware of the *HP-30b* does not provide space for variables. Thus, variable-based commands (e.g. many matrix commands) cannot be featured on your *WP 34S.*
CLx		Press [CLx].
CLΣ		Press [CLΣ].
COMB		Press [Cy,x].
[orange] COMPLEX	n/a	The hardware of the *HP-30b* does not provide space for different data types. Thus, complex numbers have to be put in two registers each on your *WP 34S* as described on p. *29*.
[orange] CONVERT	[CONV]	
CORR		Press [r].
COSH		Press [HYP] [COS].
CPXRES	n/a	See COMPLEX.
CPX?		
[orange] CUSTOM	n/a	See ASSIGN.
DECM		Press [H.d].
DEG		Press [DEG].

HP-42S	*WP 34S*	**Remarks**
DEL	n/a	Not featured. Too dangerous, in our opinion.
DELAY	DLAY	
DELR	n/a	Matrix commands: see CLV.
DIM & DIM?		
DISP	Superfluous	FIX, SCI, ENG, ALL, RDX., and RDX, are on the keyboard.
DSE		Press DSE.
E	EEX	
EDIT & EDITN	n/a	Matrix commands: see CLV.
ENG		Press ENG.
EXITALL	n/a	See ASSIGN.
FCSTX	x̂	
FCSTY	ŷ	Press ŷ.
FIX		Press FIX.
FLAGS	TEST	SF and CF are on the keyboard.
FNRM	n/a	Matrix command: see CLV.
FP		Press FP.
GAMMA	Γ	
GETKEY	KEY?	
GETM	n/a	Matrix command: see CLV.
GRAD		Press GRAD.
GROW	n/a	Matrix command: see CLV.
HEXM	BASE 16	Press 16.
I+ and I-	n/a	Matrix commands: see CLV.
INDEX		
INPUT	PROMPT	
INSR	n/a	Matrix command: see CLV.
INTEG	∫	Press ∫.
INVRT	M^{-1}	
IP		Press IP.
ISG		Press ISG.
J+ and J-	n/a	Matrix commands: see CLV.
KEYASN	n/a	See ASSIGN.
KEYG & KEYX		
LASTx	RCL L	
LBL		Press LBL.

HP-42S	*WP 34S*	**Remarks**
LCLBL	n/a	See ASSIGN. Nevertheless, your *WP 34S* provides local labels as described on p. 67.
LINΣ	Superfluous	Your WP 34S always runs in ALLΣ mode.
LIST	n/a	Use 🖶PROG instead.
LOG	LOG_{10}	Press [LG].
MAN	CF T	This print mode is startup default on your *WP 34S*.
[MATRIX]	n/a	Matrix commands: see CLV.
MAT?		
MEAN	x̄	Press [x̄].
MENU	n/a	See ASSIGN.
[MODES]	[MODE]	
MVAR	n/a	See ASSIGN.
N!	x!	Press [!].
NEWMAT	n/a	Matrix command: see CLV.
NORM	n/a	Not featured.
NOT		Press [NOT].
OCTM	BASE 8	Press [8].
OLD	n/a	Matrix command: see CLV.
ON	n/a	Not featured.
OR		Press [OR].
PERM		Press [Py,x].
[PGM.FCN]	[P.FCN]	LBL, RTN, VIEW, PSE, ISG, and DSE are on the keyboard.
PGMINT	Contained in ∫	
PGMSLV	Contained in SLV	
PI	π	
PIXEL	gSET	See AGRAPH.
POLAR	n/a	See COMPLEX.
POSA	n/a	The alpha register contains 30 characters maximum. Automatic searching would be overkill, in our opinion.
PRA	🖶α	
PRLCD	🖶PLOT	See AGRAPH.
[PRGM]	[P/R]	
[PRINT]	[P.FCN]	PRX is on the keyboard.
[PROB]	Replaced	COMB, PERM, N!, and RAN are on the keyboard. SEED is in STAT; GAM(MA) is in X.FCN.
PROFF	CF T	Press [CF] [T].

HP-42S	*WP 34S*	**Remarks**
PRON	SF T	Press [SF] [T].
PRP	🖶PROG	
PRSTK	🖶STK	
PRUSR	n/a	See CLV.
PRV		
PRX	🖶r X	Press [↑] except in alpha mode.
PRΣ	🖶Σ	
PSE		Press [PSE].
PUTM	n/a	Matrix command: see CLV.
QUIET	n/a	See BEEP.
RAD		Press [RAD].
RAN	RAN#	Press [RAN#].
RCLEL & RCLIJ	n/a	Matrix commands: see CLV.
RDX,		Press [./,] except in alpha mode.
RDX.		
REALRES	n/a	See COMPLEX.
REAL?		
RECT	Superfluous	This mode is permanent on your *WP 34S*. See COMPLEX.
RND	ROUND	Press [RND].
RNRM	n/a	Matrix command: see CLV.
ROTXY	RL, RLC, RR, and RRC	
RSUM	n/a	Matrix commands: see CLV.
R<>R		
SDEV	s	Press [s].
SF		Press [SF]
[orange] [SHOW]	[◀]	
SINH		Press [HYP] [SIN].
SIZE	REGS	
SLOPE	L.R.	
SOLVE	SLV	Press [SLV].
[orange] [SOLVER]	Contained in SLV	
SQRT	√	
[orange] [STAT]	[STAT]	MEAN, SDEV, FCSTY, and CORR are on the keyboard; the MODL setting commands are in MODE.
STOEL & STOIJ	n/a	Matrix commands: see CLV.

HP-42S	*WP 34S*	**Remarks**
STR?	n/a	See CLV.
TANH		Press [HYP][TAN].
TONE	n/a	See BEEP.
[■] [TOP.FCN]	n/a	See ASSIGN.
TRACE	SF T	Press [SF][T].
TRANS	TRANSP	
UVEC	^{C}SIGN	
VARMENU	n/a	See CLV.
VIEW		Press [VIEW].
WMEAN	$\overline{x}_w$	
WRAP	n/a	Matrix command: see CLV.
X<>		Press [x≷].
X<0? & X<Y?	x< ?	
X≤0? & X≤Y?	x≤ ?	
X=0? & X=Y?	x= ?	Press [x= ?].
X≠0? & X≠Y?	x≠ ?	Press [x≠ ?].
X≥0? & X≥Y?	x≥ ?	
X>0? & X>Y?	x> ?	
XOR		Press [XOR].
XTOA	x→α	
YINT	L.R.	Alternatively, press [0][ŷ].
[■] ∫f(x)	Contained in ∫	
ΣREG & ΣREG?	Superfluous	See *Appendix B*.
→DEC	Superfluous	Conversions are done automatically in *integer modes*.
→DEG	°→rad	Press [CONV][A][▲][XEQ].
→H.MS		Press [→][H.MS].
→HR		Press [→][H.d].
→OCT	Superfluous	Conversions are done automatically in *integer modes*.
→POL		Press [P←].
→RAD	rad→°	Press [CONV][R][XEQ].
→REC		Press [→R].
%CH	Δ%	Press [Δ%].

HP-16C

The table for the functions of the *HP-16C* is sorted following their appearance on its keyboard, starting top left. As for the *HP-42S*, only functions carrying different names on both calculators are listed. *HP-16C* functions neither mentioned here nor printed on the keyboard of your *WP 34S* are all found in the catalog X.FCN in integer modes.

HP-16C	***WP 34S***	**Remarks**
RL, RLn	RL	In X.FCN.
RR, RRn	RR	In X.FCN.
RMD	RMDR	
RLC, RLCn	RLC	In X.FCN.
RRC, RRCn	RRC	In X.FCN.
#B	nBITS	In X.FCN.
ABS	|x|	
DBL÷	DBL /	In X.FCN.
x≷(i)	Superfluous	Every register may be used for indirection.
x≷I		
SHOW HEX	→ 16	
SHOW DEC	→ 10	
SHOW OCT	→ 8	
SHOW BIN	→ 2	
B?	BS?	In TEST.
GSB	XEQ	
HEX	16	
DEC	10	
OCT	8	
BIN	2	
SF 3, CF 3	LZON, LZOFF	In MODE. Control display of leading zeros.
SF 4, CF 4	SF C, CF C	Carry.
SF 5, CF 5	SF B, CF B	Overflow.
F?	FS?	In TEST.
(i)	Superfluous	Every register may be used for indirection.
I		
CLEAR PRGM	CLP	Note there is also CLPALL.
CLEAR REG	CLREGS	In P.FCN.
CLEAR PREFIX	Superfluous	See p. 35
WINDOW	n/a	

HP-16C	*WP 34S*	**Remarks**
SET COMPL 1'S	1COMP	In MODE. Note there is also SIGNMT.
SET COMPL 2'S	2COMP	
SET COMPL UNSGN	UNSIGN	
SST	▼	
BSP	⇦	
BST	▲	
x≤y	x≤ ?	In TEST. Note there are more tests covered there.
x<0	x< ?	
x>y	x> ?	
x>0		
FLOAT	FIX	
MEM	STATUS	
STATUS		
CHS	+/-	
◄	◄(	
►	)►	
LSTX	RCL L	
x≠y	x≠ ?	
x≠0		
x=y	x= ?	
x=0		

APPENDIX G: TROUBLESHOOTING GUIDE

Overcoming Troubles in Calculator Operation

Symptoms	Possible Cause	Recommended Therapy
After entering a decimal number such as 32.4567, the display shows 2c 64 324567 d or similar.	Inadvertently entered an integer mode.	Press f H.d to return to default floating point mode.
After entering a decimal number such as 32.4567, the display shows 32 15207/3333 or similar.	Inadvertently entered a fraction mode.	Press f H.d to return to default floating point mode.
After entering a decimal number such as 32.4567, the display shows 7 StEP 013 444 or similar.	Inadvertently entered programming mode.	Press EXIT to return to run mode.
You are working in binary integer mode and the display shows un 64 "1 0 100 1011 0110 b or alike with such strange quotes top right.	Long integer number displayed	See on p. 53 how to deal with such numbers.
Display looks like RCL X L.b 0015 527 or similar. Calculation commands are not executed.	Inadvertently entered programming mode.	Press EXIT to return to run mode.
Display looks strange, but not like the patterns shown so far.	Inadvertently entered a browser or a catalog.	Press EXIT to return to previous mode and display.
All input goes to the top line.	Inadvertently entered alpha mode.	Press EXIT to return to previous mode.

Symptoms	Possible Cause	Recommended Therapy
After entering [1][.][2][3] or alike, an unwanted decimal comma shows up. How to get the requested point?	Wrong radix mark set.	Press [h][./,] . Alternatively, choose a suitable regional setting from the table on p. 39.
After entering [1][.][2][3] or alike, an unwanted decimal point is displayed instead of a comma.		
After entering a decimal number such as 3.4567, the display just shows 3. (there is a radix mark, but decimals are missing).	Wrong output format set.	Set [h][ALL] 00 to see all the decimals you entered.
Your *WP 34S* does not understand the dates or times you enter (anymore).	Wrong date or time format set.	Choose a suitable regional setting from the table on p. 39.
After entering *hours*, *minutes*, and *seconds*, only *hours* and *minutes* are displayed.	Wrong output format set.	Set [h][FIX] 4 to see the *seconds* as well.
After entering a date such as 11.112012, the year is truncated.	Wrong output format set.	Press [h][FIX] 6 to see the complete year.
In long numbers, there are unwanted points or commas every four, three, or two digits.	E3ON or SEPON is set.	If in an integer mode, press [h][MODE][S][XEQ] for SEPOFF; if in a floating point mode, press [h][MODE][E][XEQ] for E3OFF.
After converting a number such as e.g. 0.46875 into a fraction, you get . While being true, this result is not as precise as you want it.	DENMAX is too low.	Press 0 [h][MODE][D][I][▲][XEQ] to get the maximum precision in fraction mode.
After converting a number such as e.g. 32 [1/x] into a fraction, you get a result deviating from 1/32.	DENFAC or DENFIX is set.	Press [h][MODE][D][E][XEQ] to return to DENANY.

If you experience a strange situation not mentioned here, we recommend you check your display for annunciators as they are shown on p. 36.

Overcoming Flashing Troubles

Sometimes it seems the calculator just does not want to be flashed. You may have tried the procedure described in *Appendix A* repeatedly, but the display stubbornly stays blank. Do not despair[79] – the following procedure will help:

1. If there is still a cable linking calculator and computer, remove it from both devices a.s.a.p.

2. Check the batteries.
   a. Open the battery door and remove both batteries.
   b. Check their voltage. Both batteries shall produce 2.8 … 3.2 V. If it is less, replace both of them with fresh coin cells type CR 2032.
   c. Insert the batteries properly – with their positive poles up – like pictured here. Now your calculator is well prepared. Leave the battery door open.

3. Check your computer interface and connecting cable.
   a. You have either (i) a programming cable or (ii) a micro USB cable. The latter requires a custom board installed in your calculator – if this installation was done and verified according to *Appendix H* you can be confident it is ok.
   b. Check both sides of your cable: are there any bent or dirty pins? If not and the cable also shows no signs of being cut or torn, it should be ok. If dubious, use another cable of the same kind.
   c. Connect your chosen cable to your computer. For case (i), use a suitable converter if required (see p. 147). Using such a converter with a PC, you shall see it listed in <System Control> <Device Manager> as a "*USB Serial Converter*" (remember its port) – else this converter must be replaced.
   d. Check the contacts at your calculator being dirt-free. Connect the cable to your calculator. For case (ii) and using a PC, you shall see it listed in <System Control> <Device Manager> as a "*FT230X USB Half UART*" (remember its port) – else there is a problem with the cable or the custom board (see p. 197).

4. Now all the hardware is well prepared. Start *MySamBa* on your computer and enter the port (as determined in 3.c or 3.d) and the file information (see p. 147). Do not start sending yet, just leave the window open!

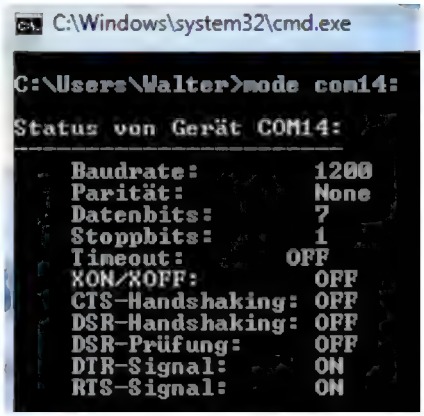
[79] No such calculator ended as a paperweight just by flashing yet (but many batteries were drained for sure by programming cables staying connected for too long and / or by calculators quietly continuing running full speed).

5. Focus on your calculator again – it is connected to your computer now. Press [ON] firmly for two seconds.

   Does the calculator display react now? Fine – continue with step 4 on p. *149* for first flashing, case (i), or on p. *150* for case (ii), or on p. *152* for updating your *WP 34S*.

   If the display stays blank, however, press RESET and continue with the next step here.

6. Verify the port is known to the operating system. On a PC, open the start window and enter CMD – this will open the good old *DOS* window; therein type MODE COM***x*** with ***x*** being the port number as determined in 3.c or 3.d. MODE should return data like these here or the equivalent in your language. Do not care for particular values since the parameters will be controlled by *MySamBa* anyway.

   If you get a similar message, continue with step 3 on p. *149* for case (i), or on p. *150* for case (ii).

   If the port is unknown, however, return to step 3.c above and try another one.

Two of our early *HP-20b*-based calculators (without USB boards built in) developed broken serial input lines in service. These lines cannot be repaired easily. Those are the only cases we know so far a *WP 34S* showing a blank display became unusable since it cannot be flashed anymore.

APPENDIX H: ADDITIONAL INFORMATION FOR ADVANCED USERS

The information provided here may ease your work when you are going to do advanced computations, especially programming. This section also covers optional hardware modifications (see p. 192). All the topics collected here require special care and/or a deeper understanding of the respective 'mechanics' of the *WP 34S* – else you might be surprised by the consequences. **Use the information provided in this appendix at your own risk!**

Changing Word Size in Integer Modes

Increasing or reducing the word size in integer modes will affect the current stack contents only (see the description of WSIZE in the *IOP*). All other memory content of your *WP 34S* will stay as is. This differs from the implementation as known from the *HP-16C*, where all registers were remapped on word size changes.

See the following **example** starting with your *WP 34S* in BASE 16, WSIZE 16, SSIZE 4, and with the stack filled with F12E:

I	F12E	F12E	F12E	F12E	F12E	F12E	F12E
L	old stuff	old stuff	E61	61	61	61	2E
T	F12E	F12E	F12E	2E	2E	2E	2E
Z	F12E	F12E	F12E	2E	2E	2E	2E
Y	F12E	F12E	F12E	2E	8F	8F	2E
X	F12E	E61	FF8F	8F	2E	2E	1962
Input	STO I	E61	+	WSIZE 8	RCL I	WSIZE 16	×

ATTENTION: Note that increasing the word size will just add empty bits to the significant side. Thus, a negative integer will immediately become positive then. There is no automatic sign extension! If you want it, you will have to take care of it yourself.

Mode Storing and Recalling

The command STOM stores mode data in a register. The following table shows more details. Types are coded G for general, F for fractions, D for decimal, and I for integer.

Bits	Type	Contents
0 … 3	G	*LCD* contrast setting (0 … 7, always 4 for the emulator)
4, 5	F	Fractions denominator mode (DENANY = 0, DENFAC = 1, DENFIX = 2)
6 … 19	F	DENMAX (14 bits for 0 … 9999)
20	F	Clear for PROFRC, set for IMPFRC
21	F	Set if fraction mode is on

Bits	Type	Contents
22, 23	D	Decimal display mode (ALL = 0, FIX = 1, SCI = 2, ENG = 3)
24 … 27	D	Number of decimals (4 bits for 0 … 11)
28	D	Clear for SCIOVR, set for ENGOVR
29	D	Clear for RDX. – set for RDX,
30	D	Set if E3OFF
31	I	Set if SEPOFF
32	I	Set if an integer mode is on
33	I	Set if LZON
34, 35	I	Integer sign mode (1COMPL = 1, 2COMPL = 0, UNSIGN = 2, SIGNMT = 3)
36 … 39	I	Integer base (4 bits for 2 … 16 coded as 1 … 15)
40 … 45	I	WSIZE (6 bits for 1 … 64 coded as 0 … 63)
46	D	Set if DBLON
47	D	Time format (24h = 0, 12h = 1)
48, 49	G	Print mode (0 … 3, see 🖶MODE)
50		Not used yet
51	G	Stack size (SSIZE4 = 0, SSIZE8 = 1)
52, 53	D	Date format (Y.MD = 1, M.DY = 2, D.MY = 3)
54, 55	D	Angular mode (DEG = 0 ,RAD = 1, GRAD = 2)
56 … 58	D	Curve fit model (LINF = 0, EXPF = 1, POWERF = 2, LOGF = 3, BESTF = 4)
59	G	Clear for FAST, set for SLOW (always clear on the emulator)
60 … 62	D	Rounding mode (0 … 7, see RM)
63	D	Clear for JG1782, set for JG1582

Commands for Advanced Users

There are three ON combinations not documented above:

[ON] + [C] : Tells the system a quartz **c**rystal is installed for an accurate real time clock. Find more about this **hardware modification** on p. 192ff. The quartz crystal is an inevitable prerequisite for the clock being useful in medium to long range (see DATE, TIME, STOPW, SETDAT, SETTIM), and it is required for print operations as well. The system asks `Crystal? InSTALLEd` – keep [ON] held down and press [C] a second time to activate the crystal. [ON] + [C] will work only if the crystal was not activated yet.

[ON] + [C] should be only necessary if you have loaded the file `calc.bin` or `calc_full.bin`, while the other files enable the crystal automatically.

WARNING: If the crystal is not installed, the system will hang if [ON] + [C] is entered or if a firmware is booted requiring the crystal, and

a hard reset (actuating the RESET button) or a battery pull will be required!

Lack of the *IR* diode cannot cause a system hang since it cannot be checked automatically. Note this diode may alternatively be connected using one of the small *USB* boards mentioned in *Appendix A*. Find an installation guide for such a board (version 2) at the end of this appendix *here*.

[ON] + [D] : Toggles **d**ebugging mode, echoed by `Debug ON` or `Debug OFF`. The small = annunciator is lit for debug mode set. Note your *WP 34S* will not shut down automatically in this mode. [ON] + [D] is used in the updating procedure on p. *152* (followed there by [ON] + [S]) but was not explained there.

[ON] + [S] Stands for *My**S**amBa*: the system asks `SAM-BA? boot` – keep [ON] held down and press [S] a second time to clear the GPNVM1 bit and turn the calculator off. This will work in debugging mode only (see above) to prevent accidental access to this potentially dangerous feature. It is used in the updating procedure on p. *152* but was not explained there.

WARNING: After issuing this combination, you can only boot in *MySamBa* boot mode! Without the *MySamBa* software and a proper cable (as mentioned on p. *147*), you are lost!

Double Precision (*DP*) Calculations and Mode Switching

Your *WP 34S* starts in *SP* mode per default, wherein 16 digit precision is reached in all calculations. Switching between *SP* and integer modes was discussed on p. *159*. Additionally, you may use your *WP 34S* in *DP* mode. Each *DP* register will contain 16 *bytes* instead of eight, allowing for 34 digits instead of 16 (see below). Note matrix commands will not work in *DP*.

DP allows for more precise calculations. While some computations will reach high accuracy, **we do not warrant 34 digit precision in all calculations in *DP* mode.**

The following figure illustrates what happens in memory in transitions between *SP* and *DP* modes, assuming startup in *SP* mode with REGS 16. *RRA* stands for 'relative register address'.

Absolute address	Startup memory allocation		After executing DBLON		Then after DBLOFF	
	Contents	*RRA*	Contents	*RRA*	Contents	*RRA*
X+11	*k = 1.40E-397*	R111 = K	*1.40E-397*	K = R111	*1.40E-397*	K = R111
X+10	…	…			…	…
…	…	…	…	…	…	…
X+1	*y = -33.8*	R101 = Y	…	C = R106	*-33.8*	Y = R101
X	*x = 123.0*	R100 = X			*123.0*	X = R100
X-1	*r15 = -43.6*	R15	…	B	*0.0*	R15
X-2	*r14 = 167.9*	R14			*0.0*	R14
X-3	…	R13	…	A	*0.0*	R13
X-4	…	R12			*0.0*	R12
X-5	…	R11	…	T	*0.0*	R11
X-6	…	R10			*0.0*	R10
X-7	…	R09	…	Z	*0.0*	R09
X-8	…	R08			*0.0*	R08
X-9	…	R07	*-33.8*	Y = R101	*0.0*	R07
X-10	…	R06			*0.0*	R06
X-11	…	R05	*123.0*	X = R100	*0.0*	R05
X-12	*r04 = -12.9*	R04			*0.0*	R04
X-13	*r03 = -1234.89*	R03	*-1.95E184*	R01	*-1234.89*	R03
X-14	*r02 = 5.43E-396*	R02			*5.43E-396*	R02
X-15	*r01 = 6.6*	R01	*1.03E182*	R00	*6.6*	R01
X-16	*r00 = 0.54*	R00			*0.54*	R00
X-17						

Going from *SP* to *DP* mode via DBLON, the contents of the twelve registers **X … K** are copied, cutting 48 *bytes* into the former *SP* numbered register sector. So the top twelve *SP* numbered registers will be lost in such a transition. All other memory contents stay where and as they were – just each *DP RRA* covers what were two *SP* registers before. The space allocated for summation registers will not change in such transitions.

Starting with the default memory configuration and executing DBLON then will leave you with 44 *DP* registers. Executing REGS with an argument >44 in *DP* is legal, but the sector of global numbered registers will then cut into the former program sector.

Returning from *DP* to *SP* mode, the lettered registers are copied again. Everything else stays where and as it was, if you used ≤44 *DP* registers – just each *SP RRA* points to only one half of a former *DP* register; and the memory released by the shrinking special registers allows for adding (or returning) twelve numbered registers on top, each now containing zero.

With >44 *DP* registers, the correspondence becomes more complicated – the number of global registers will not, however, exceed 112.

For the following table, assume startup in BASE 10, WSIZE 32, REGS 16. Now see the contents of **J**, **K**, and the lowest numbered registers, checked by recalling them to **X**:

	J	K	R00	R01	R02	R03
Starts with e.g.	3.504 d	14 d	54 d	66 d	543 d	126.441 d
DECM	3.43^{-395}	1.40^{-397}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
DBLON[80]	3.43^{-395}	1.40^{-397}	1.03^{-HIG}	1.95^{-HIG}	n/a	n/a
Recall by sRCL	1.40^{-397}	1.28^{-23}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
… and by iRCL	14.00	0.00	54.00	66.00	543.00	126.441.00
DBLOFF	3.43^{-395}	1.40^{-397}	5.40^{-397}	6.60^{-397}	5.43^{-396}	1.23^{-393}
Recall by dRCL	Out of range Error	Out of range Error	0.00	0.00	n/a	n/a
DBLON	3.43^{-395}	1.40^{-397}	1.03^{-HIG}	1.95^{-HIG}	n/a	n/a
RCL J, STO J, 123E456 STO 00, then recall by sRCL	1.40^{-397}	1.28^{-23}	1.23^{-396}	5.12^{-30}	5.43^{-396}	1.23^{-393}
… and by iRCL	14.00	0.00	123.00	0.00	543.00	126.441.00
DBLOFF	3.43^{-395}	1.40^{-397}	1.23^{-396}	5.12^{-30}	5.43^{-396}	1.23^{-393}
Recall by dRCL	Out of range Error	Out of range Error	+∞ Error	0.00	n/a	n/a

Note iRCL and sRCL keep working as explained on p. 159.

In *DP* mode, ◀ shows the first (most significant) 16 digits of the 34-digit mantissa of ***x*** and its four digit exponent, and ▶ displays the 18 trailing digits, both as *temporary messages*. For **example**, π ◀ returns here

, and ▶

.

[80] *DP* mode reals are stored coarsely following *decimal128* packed coding, though with some exceptions. The lowest 110 bits take the rightmost 33 digits of the *significand*. Going left, a 12 bit exponent field follows, then 5 bits used and coded exactly as in *SP*, and finally the sign bit. The greatest value of the stored exponent is $10\ 1111\ 1111\ 1111_2 = 12287_{10}$. For reasons like the ones explained for *SP*, 6176 must be subtracted from this value to get the true exponent of the floating point number represented. Thus, *DP* supports 34-digit numbers within $10^{-6143} \le |x| < 10^{+6145}$. Coding works in full analogy to the way described for *SP* in App. B.
Even smaller numbers may be entered using a decimal mantissa, but you will lose one digit per factor of 10. The same happens if you divide 10^{-6143} by 10 several times. At 10^{-6176}, only one digit will be left, stored as hexadecimal 1. Divide it by 1.999 999 999 99 and the result will remain 10^{-6176}. Divide it by 2 instead and the result will become zero.
Numbers beyond the interval $10^{-999} \le |x| < 10^{+1000}$ will be displayed with -HIG or HIG in their exponent, respectively. Only ◀ will show you the true exponent of them. This way ***r00*** = 1.032 000 000 000 000 054 E-6155 and ***r01*** = 1.951 656 000 000 000 000 000 543 E-6152 are found here.
Returning to *SP* with numbers in lettered registers exceeding the *SP* number range will cause 0 or InFinitY being displayed instead.

Remember not every return may be as precise as this one. And errors accumulate as explained in the footnote at *CONST*.

ATTENTION: Rounding mode (see RM) may affect the results of *DP* calculations!

Further Commands Used in Library and *XROM* Routines

The operations listed below are used by the programmers of said routines. They ease their work by allowing some more comfort in writing programs than the original function set. These commands and pseudo-commands are explained here to foster understanding of those routines, but they are not accessible through any catalog. The assembler (often already its preprocessor) will translate most of them into proper program steps employing the commands documented above. See the *WP 34s Assembler Tool Suite User Guide* for additional information.

(Pseudo-) Command	*XROM* or *LIB*	Function
A..D→	*XROM*	saves a, b, c, and d at a volatile temporary storage. Volatile means it will vanish some 500ms after last keystroke.
GSB ***longlabel***	Both	This pseudo-command allows for calling long alphabetic labels. The assembler translates GSB into BSRB or BSRF (see p. 191) – or into XEQ (via a jump table in *XROM* or via a label in *LIB*) if the distance is too far. **ATTENTION:** Note that STOP and PROMPT will not work in *XROM* on your *WP 34S*. And there are no LBL nor END statements in *XROM* code. Long alphabetic labels (looking like ***longlabel*::**) may show up in code written for the assembler but will be resolved.
JMP ***longlabel***	Both	This pseudo-command allows for branching to long alphabetic labels. The assembler translates JMP into BACK or SKIP – or GTO (via a jump table in *XROM* or via a label in *LIB*) if the distance is too far.
Num ***sn***	*XROM*	with ***sn*** = 0, π, √2π, … inserts the respective constant like recalling it from CONST would do.
POPUSR	*XROM*	See XEQ or XEQUSR.
XEQ ***label***	*XROM*	calls the user routine carrying the global label specified (there are no LBL statements in *XROM*). XEQ'xyz' in *XROM* must be followed by POPUSR immediately to restore the *XROM* execution state (registers, flags, return addresses) correctly. See also xIN.

(Pseudo-) Command	*XROM* or *LIB*	Function
XEQUSR ***label***	*XROM*	calls the user routine containing the function to be solved, integrated, summed, or multiplied, respectively. The label of this function is transmitted as a parameter of the respective user command described above (such as e.g. SLV). XEQUSR must be followed by POPUSR immediately to restore the *XROM* execution state (registers, flags, return addresses) correctly.
xIN ***type***	*XROM*	with ***type*** = `NILADIC`, `MONADIC`, `DYADIC`, `TRIADIC`, or `..._COMPLEX` defines how many stack levels are used for parameter input to the function under consideration. Furthermore it does some initialization work (e.g. SSIZE8 and DBLON). xIN is the recommended way to start an *XROM* routine. Thereafter, SSIZE4 is legal but DBLOFF is not. Note xIN cannot nest and *XROM* routines using xIN cannot call user code.
XLBL"***label***"	*XROM*	defines the eXternal LaBeL of this routine. XLBL doesn't generate any code, it provides an entry point that the C function tables can take advantage of. Typically, ***label*** equals the respective command name.
xOUT ***way***	*XROM*	typically, ***way*** = `xOUT_NORMAL` . xOUT cleans and reverts the settings of xIN, taking care of a proper return including the correct setting of l and the stack. Generally, xOUT shall be the last command of an *XROM* routine.
_INT ***n***	*XROM*	1, 2, …, 128, but also calculated constants are possible. _INT inserts the respective constant like # does.
"***text***"	Both	The double quotes allow for convenient entry of text strings. The assembler translates the string into the amount of α-statements required.
→A..D	*XROM*	recalls a, b, c, and d from their temporary storage (very short range, see above).

Furthermore, there are several 'alias' spellings to ease input of some commands or even of individual nonstandard characters on an English computer keyboard. E.g. the Greek letter 'α' is often replaced by a Latin 'a', the print character 🖶 by 'P.' etc. Again, the assembler will take care of translating the aliases.

See the file `8queens_alias.wp34s` (or other files of type `…_alias.wp34s` in 'library') for examples. `Command-Aliases.pdf` in 'doc' contains a list of all aliases used (some 50 pages).

Assembler Output

The Assembler takes your .wp34s text file as input and generates optimized code for your *WP 34S*. It does not generate optimized code for your further editing nor maintaining your routines. "Optimized" means it will use as little of the limited resources of your *WP 34S* in the programs as possible, less labels in particular. Thus expect heavy use of BACK and SKIP instead of GTO in assembled programs since all required information for this conversion is available at assembly time.

Furthermore, there are two special commands frequently replacing GSB / XEQ in assembled code:

1. BSRB ***n*** calls a subroutine starting ***n*** steps backwards with $0 \le n \le 255$. It pushes the program counter on the subroutine return stack and executes BACK ***n*** then.
2. BSRF ***n*** calls a subroutine starting ***n*** steps forwards. It pushes the program counter on the subroutine return stack and executes SKIP ***n*** then.

The subroutines called this way do not require a starting label. So, BSRB and BSRF are useful if you are short on local labels – as BACK, SKIP, and CASE are. On the other hand, if you edit a section of your routine that is crossed by one or more BSRB, BSRF, BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually. We recommend editing unassembled code and leaving the generation of label-less branches to the assembler for sake of readability, maintainability, and reliability of your routines.

Basic Electrical Hardware Specifications of the *HP-20b* and *HP-30b*

Power supply: $3V$ (2 CR2032 coin cells)

Processor: Atmel AT91SAM7L128

- ARM 7 core with thumb mode ($32kHz$ to $40MHz$)
- RAM: $2kB$ with backup, $4kB$ volatile
- ROM: $128kB$ flash, SAM-BA boot loader
- I/O: parallel ports, serial port for flashing (see picture below for pinout)
- *LCD* controller
- Elaborate power management
- Multiple clock sources (RC, quartz, PLL), real time clock

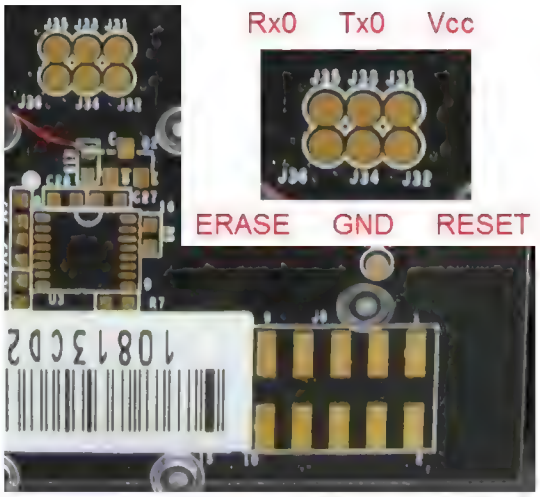
Liquid crystal display with 400 segments:

- 12 x 9 segments (mantissa), 3 x 7 segments (exponent), 2 signs,
- 6 x 43 pixels in a dot matrix,
- 10 annunciators, and
- 1 additional fixed symbol (see p. 35).

Keyboard with 37 keys in 5 to 6 columns by 7 rows

The six pad serial port of the calculator is located under the battery door, below the RESET hole. For those who want to connect to it directly, here is the pinout. It is identical for both business calculator models, the *HP-20b* and the *HP-30b*. The pitch is $2mm$. The picture is of the printed circuit board uncovered, the RESET button is north of the six pads.

There is an additional JTAG connector at the southern end of the main board (not accessible from outside), partially covered by a tag here.



Tuning the Hardware of Your *WP 34S*

As mentioned at the very beginning of this manual, there are three optional hardware modifications you may benefit from. Please read the respective descriptions and decide what you are interested in. Then read the complete instructions for these cases before actually starting the modification – this is to prevent you will end with your *WP 34S* half usable. These are the opportunities:

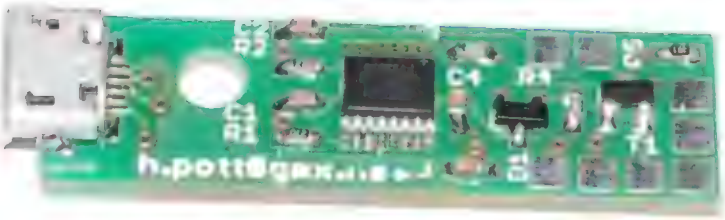
1. Create an accurate real-time clock by adding a quartz crystal and two capacitors on the main board of your *WP 34S*.

   This modification requires some fine soldering skills due to the size of the capacitors – they are SMD components and significantly smaller than a grain of rice! Besides fixing these two properly at the correct location, this modification is an easy job. See *Alexander Oestert's* very nice instruction file at *wp34s.svn.sourceforge.net/viewvc/wp34s/doc/How_to_install_crystal_and_IR_diode.pdf* .

2. Establish an *IR* line for printing on an HP-82240A/B by adding an *IR* diode and a resistor to the package after modification 1. Electrically, this part is significantly easier than the job above. There is some additional mechanical work: drilling a little hole in the plastic case, requiring a slow hand and a correct drilling bit. The complete job is also described in the file mentioned above.

3. You may combine the setup of both the *IR* line to the printer and an *USB* connection for data exchange and power supply at once by installing a little custom board *Harald Pott* developed [81] (it was already mentioned in *Appendix A*).

   Here you will need some soldering skills as well (not as fine as for modification 1 above, but close). And mechanical skills are required, too, since an extra gap in the calculator case for the micro



[81] *Harald* developed more than this custom board for your *WP 34S*. We chose what he calls PCB version 2. We rate it most helpful since it combines the features mentioned above and is easy to install as described below. See also *http://wp34s.svn.sourceforge.net/viewvc/wp34s/doc/WP_34S_USB_installation.pdf* for information about his other boards and *Harald's* address.

USB socket is needed in addition to the little hole for the IR diode.

In the following, the job is described for the board as pictured above and an *HP-30b*, based on material provided by *Alexander* and *Harald*:

a. **Open your calculator (the easy part).**
Tools needed: Small Philips screwdriver, wooden toothpick.

Procedure:

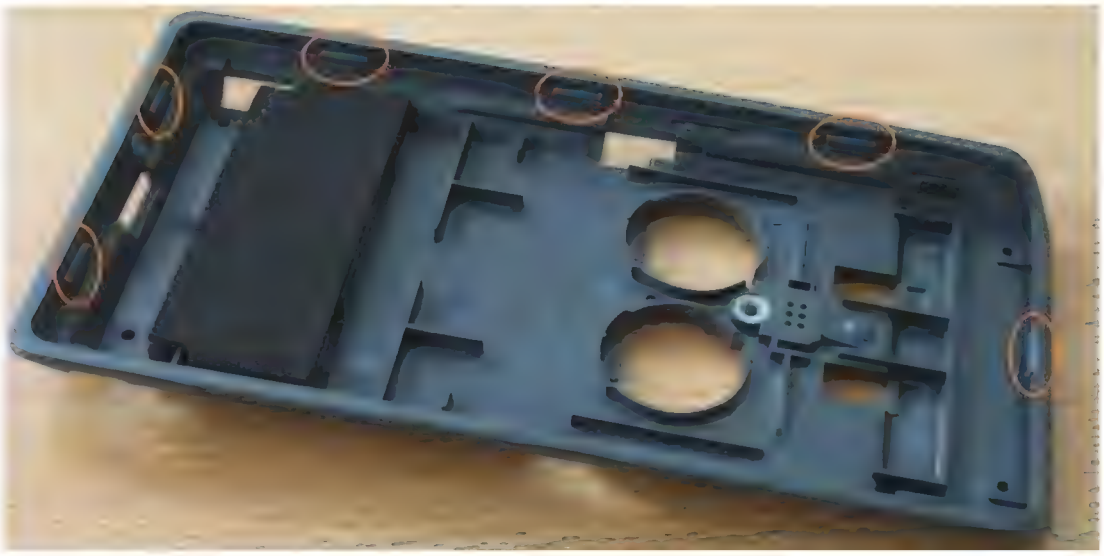
Remove the battery cover by sliding it open.

Remove the two coin cells using the toothpick.

Remove the three screws you see. Two more are hidden under each end of the bottom rubber foot – remove these screws, too.

b. **Open your calculator (the hard part).**
Tool needed: Swiss Army knife.

Helpful information: The shiny front part of the *HP-30b* is clipped into the grey frame. There are three clips at either side, two at the top, one at the bottom. See the picture of an opened frame here:



Procedure:

Carefully insert the big blade of the knife between front part and frame near the center of one side. Force the shiny part up.

Repeat on the opposite side.

Move the blade toward the calculator top. Gently force the shiny part up on both sides. Think of the *LCD*, so be careful here!

Insert the blade between front part and frame at calculator top. Gently force the shiny part up. The top of the shiny part should be loose now.

Finally, get the bottom part loose the same way.

Turn the shiny part over (printed circuit board up) and lay it on your desk. Put the grey frame aside.

c. **Prepare the mount for the *USB* board.**
Tool needed: Small knife with a sharp thin blade.

Procedure:

Cut away the plastic supports of the top right screw post and of the *LCD* frame in that area. The bottom picture shows how it shall look when you are done.

d. **Mount the *USB* board.**
Material needed: Arbitrary plastic glue, e.g. *UHU*.

Procedure:

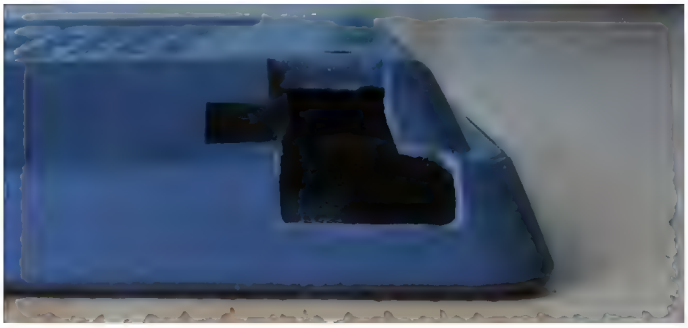
Fix the board around the screw post, left to "h.pott", and below the terminal end. Then lay the assembly aside for drying.

e. **Make the opening for the *USB* plug.**
Tool needed: Small square file.

Procedure:

Take the grey frame and lay it on the desk bottom up, rubber foot pointing to you. Then you need to extend the opening top right.

Take measures looking at the mounted *USB* board. The pictures show how the frame shall look when you are done. Remember the battery door covers the upper part.



f. **Connect the *USB* board to the *IR* diode.**

Tool needed: Soldering iron (tip shall not significantly exceed the size of the smallest terminal you want to solder, < 30W recommended).

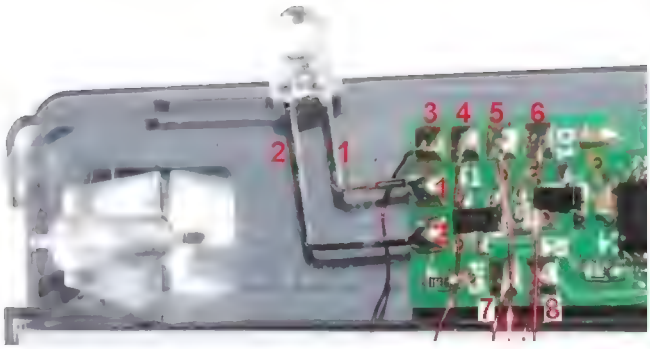
Material needed: Solder.

Procedure:

Heat up the soldering iron.

Look for a decent place where the diode may peek through the grey frame. Stay away from the clips.

Turn the diode so its longer leg will be at right. Bend that leg so it will reach terminal 1 (see the picture). Bend the shorter leg so it will reach terminal 2. Cut the legs at proper length. Both legs must not contact each other.



Apply a tiny bit of solder on terminal 1. Do the same for terminal 2 and for the ends of both legs of the diode.

Solder the formerly longer diode leg to terminal 1. Solder the other one to terminal 2.

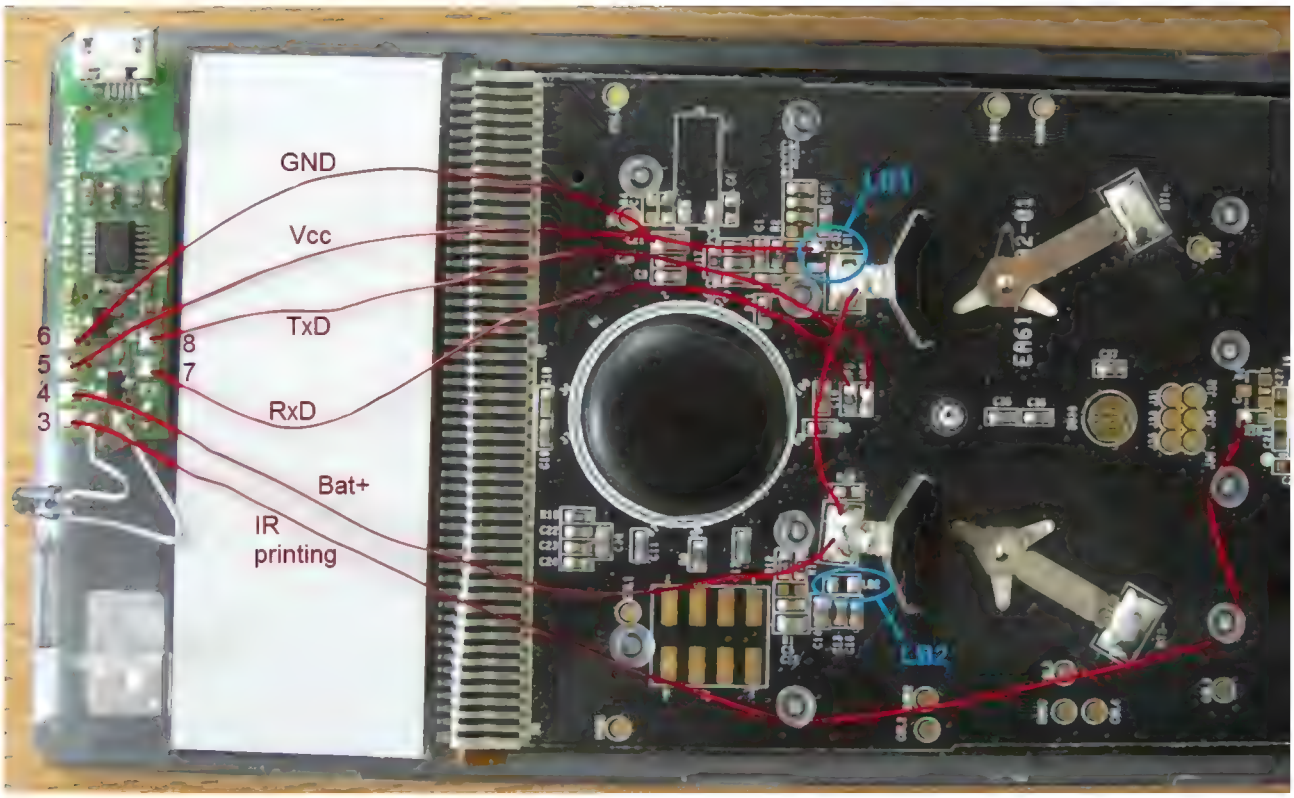
g. **Connect the *USB* board to the main board.**

Tool needed: Soldering iron as for step f.

Material needed: Isolated wire (diameter about 0.3mm to be easily laid out), solder.

Procedure:

There are six connections between both boards and one more on the main board.



First, however, two components must be removed from the main board:

Remove LB1. Since you are going to discard it anyway, put the tip of the heated soldering iron with a bit of solder on both terminals of component LB1 and heat it up. After a while, the solder below LB1 shall melt, so LB1 can be moved easily. It may even stick to the tip of the soldering iron and may be lifted this way.

Remove LB2 the same way.

Connect the terminals as listed, i.e. number

- 3 (IR printing) to upper terminal of R18,
- 4 (Bat+) to the big upper battery terminal,
- 5 (Vcc) to the right terminal of LB1,
- 6 (GND) to the top end of C21,
- 7 (RxD) to either end of JP5,[82]
- 8 (TxD) to either end of JP6,
- and connect both big upper battery terminals.

For each connection, cut a piece of wire (some 150mm). Skin its first 2mm, put some solder on it, put some solder on the respective terminals on either board, and solder one end to the terminal on the *USB* board. Then determine the necessary length for an easy connection, cut the wire to proper length, skin its last 2mm, and put some solder on it. Solder this end to the terminal on the main board. Finally, check the proper connection using an Ohmmeter (test voltage = some 1.5V).[83]

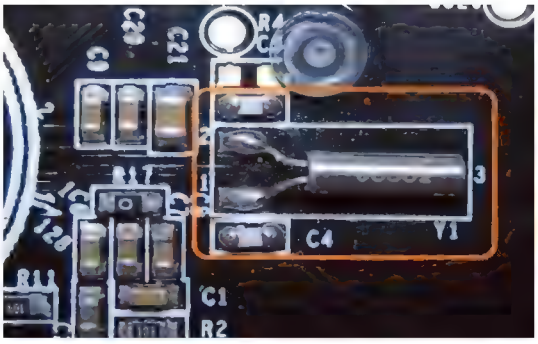
h. **Optional: Install the additional components for an accurate real-time clock.**

Tools needed: Soldering iron as for step f, wooden toothpick for holding components.

Material needed: Two 18pF 0603 capacitors, one 32.768kHz quartz crystal, solder.

Procedure:

Check top right of the main board. If the quartz crystal Y1 and the two capacitors C3 and C4 next to it are not installed yet, you can do that now. It is not mandatory, but will help in communication (cf. p. 166). It is inevitably required, however, for using the full set of print, time, and date commands including the stopwatch. See the picture here for comparison and the file mentioned at point 1 *above* for parts specifications and work instructions.



Soldering these capacitors is a challenge for the common man since they are really (really!) small. Alternatively, you may use wired capacitors: the result does not look half as professional but it is far easier achieved and does the job as well.

Remember to activate the crystal when the tuning is completed (see p. 185).

[82] JP5 and JP6 are jumpers, i.e. 0Ω resistors.

[83] Since the electronic components are small and densely packed on these boards, there is a fair chance of producing solder bridges – thoroughly check and remove where you produced them. Also take care that you do not heat any mounted component to an extent where it will start moving away. Soldering on the main board requires a slow hand, good eyesight, and a bit of experience.

i. **Drill the hole for the *IR* diode.**

Tool needed: Drill bit matching the size of the diode, but also any other tool making decent round holes of suitable size may do (even a Swiss Army knife), if you proceed slowly.

Procedure:

Hold the grey frame over the mounted diode to learn the location where to drill the hole. Its center shall be about $5mm$ away from the edge of the frame. Drill the hole there.

j. **Reassemble.**

Tool needed: small Philips screwdriver.

Procedure:

Check your shiny setup will properly fit. Then insert it in the grey frame with the diode pointing in its hole. Press to click into place (best do top and bottom first – do not press on the *LCD* window nor apply exaggerated force!).

Turn in the five screws.

Return the coin cells.

Close the battery door – your calculator is done!

k. **Connect your calculator to a *USB* port of your computer.**

Material needed: micro *USB* cable.

Procedure:

Plug in the cable in your calculator. Depending on the shape of the plug, a little cutting may be required on the plastic part of the plug to allow for proper insertion.

Turn on your calculator.

Connect it to your computer. If you did everything right, the operating system of your *PC* will automatically start searching for a driver on connection. Most probably it will not find any. Get the newest drivers for your operating system from *FTDI* for free – look here: *http://www.ftdichip.com/Drivers/VCP.htm* . After driver installation, your calculator will show up in the *Device Manager* as a '*FT230X USB Half UART*' when connected, and a COM port will be allocated. You will need this COM port for flashing.

Now you are prepared for easy flashing and updating as described in *Appendix A*.

APPENDIX I: ADVANCED MATHEMATICAL FUNCTIONS

Your *WP 34S* contains several operations covering advanced mathematics. They are all implemented for the first time on an *RPN* calculator; and all work in DECM. Find those functions collected here and described in more detail than in the *IOP*, together with a few traditional pocket calculator functions matching the topic.

For reasons explained in the first section of this manual, we assume you are able to read and understand mathematical formulas for real domain functions. Wherever complex numbers may be valid input or output, be sure you understand the respective basic mathematical concepts (at minimum as outlined on p. 29). Else leave these functions aside. By experience, it is not beneficial to use something you do neither overview nor know the background of – it may even become dangerous for you and your fellow men.

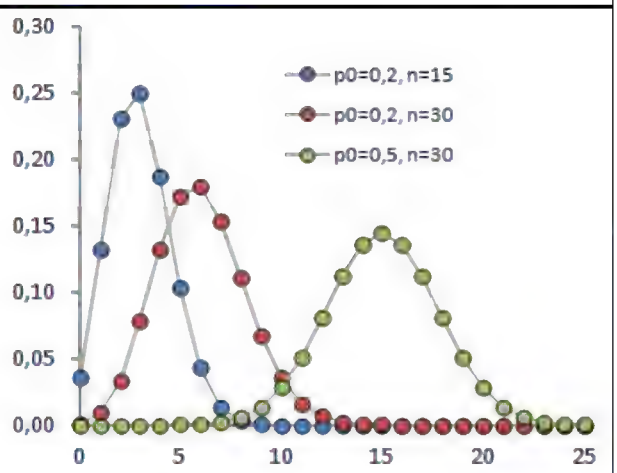
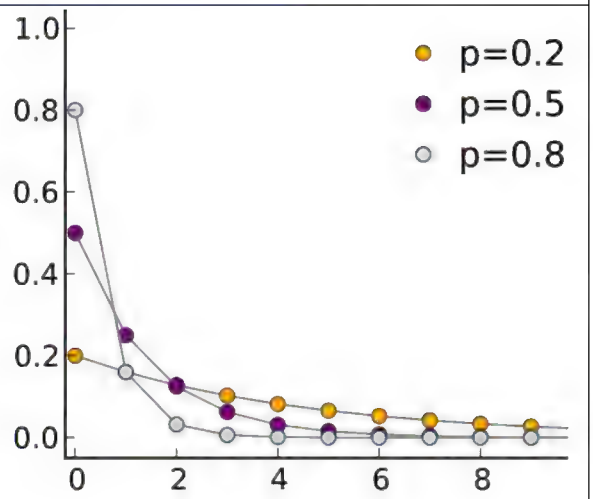
Numbers

The following are all one-number functions unless specified otherwise.

Name	Remarks (see p. 74 for general information)
B_n	B_n returns the Bernoulli number for an integer $\boldsymbol{n} > 0$ given in **X**: $B_n = (-1)^{n+1} n \cdot \zeta(1-n)$.
B_n^*	B_n^* works with the old definition instead: $B_n^* = \frac{2 \cdot (2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$. See p. 208 for ζ(x).
COMB	(2) The number of *combinations* is $C_{y,x} = \binom{y}{x} = \frac{y!}{x! \cdot (y-x)!}$.
FIB	In integer modes, returns the Fibonacci number f_n with $\boldsymbol{n} = \boldsymbol{x}$. These numbers are defined as $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $\boldsymbol{n} \geq 2$. With UNSIGN, f_{93} is the maximum before an overflow occurs.
	In DECM, returns the extended Fibonacci number $$F_x = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^x - \left(\frac{2}{1+\sqrt{5}}\right)^x \cos(x\pi)\right] = \frac{1}{\sqrt{5}}\left[\Phi^x - \Phi^{-x}\cos(x\pi)\right]$$ for an arbitrary real number $\boldsymbol{x}$, with Φ denoting the golden ratio.
PERM	(2) The number of *permutations* is $P_{y,x} = \frac{y!}{(y-x)!} = x! \cdot C_{y,x}$.

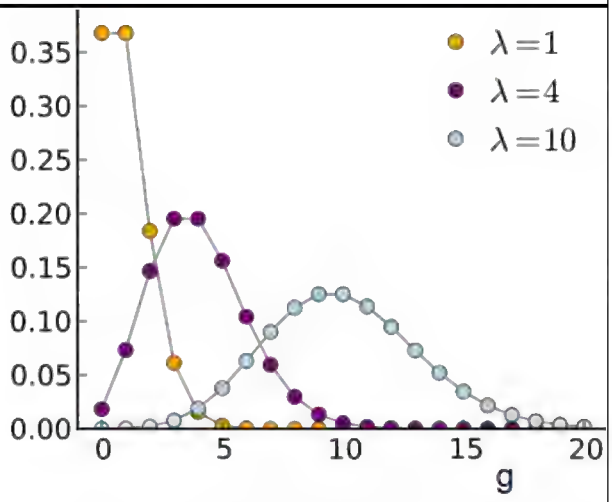
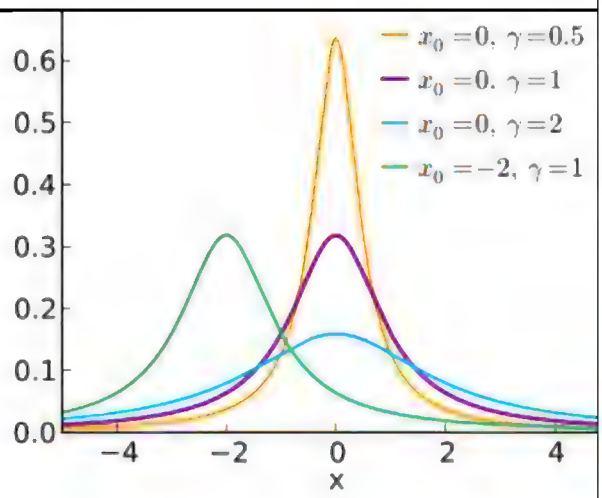
Statistical Distributions

Stack-wise, the following are all one-number functions. They are all stored in PROB. In the table below, the three discrete distributions are covered first, the continuous ones thereafter. Typical plots are shown for the *pmf*s or the *pdf*s, respectively.

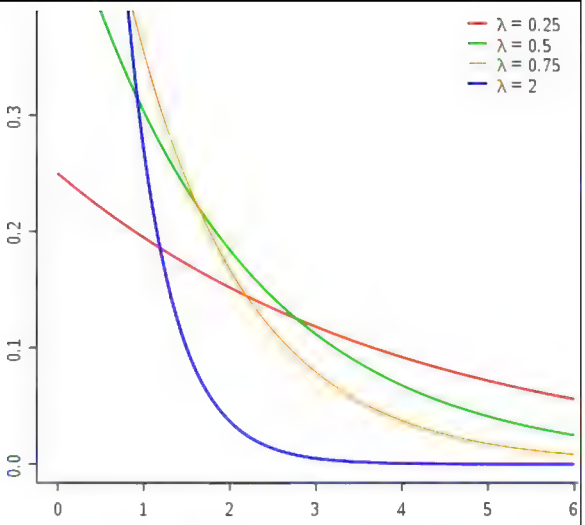
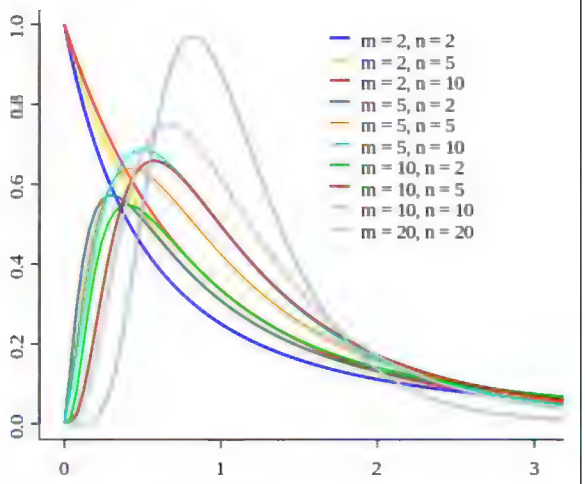
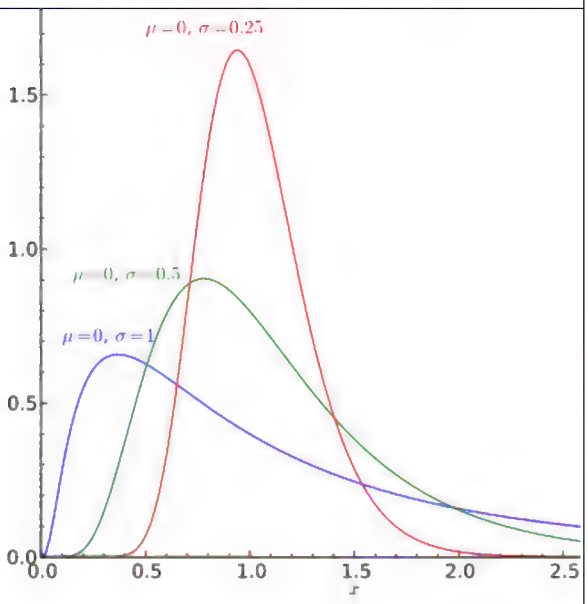
Name	Remarks (see p. 74 for general information)
Binom	*Binomial distribution* with the *number of successes* ***g*** in **X**, the *gross probability of a success* $\boldsymbol{p_0}$ in **J** and the *sample size* ***n*** in **K**. Binom$_P$ returns $p_B(g;n;p_0)=\binom{n}{g}\cdot p_0^g\cdot(1-p_0)^{n-g}$ $=C_{n,g}\cdot p_0^g\cdot(1-p_0)^{n-g}$ (see COMB *above*).  Binom returns $F_B(m;n;p_0)=\sum_{g=0}^{m}p_B(g;n;p_0)$ with the maximum number of successes ***m*** in **X**.[84] The *binomial distribution* is fundamental for error statistics in industrial sampling, e.g. for designing test plans. If you want to know, for **example**, the probability for finding no faulty items in a sample of fifteen items drawn from a batch of 300 wherein you expect 3% defective items overall, this will tell you: 0.03 [STO][J] 15 [STO][K] 0 **Binom** returns 0.633 – so the odds are almost two out of three that you will not detect any defect in your sample! [85] Read here for more information: *http://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm* .
Geom	*Geometric distribution*: Geom$_P$ returns $f_{Ge}(n)=p_0(1-p_0)^n$, Geom returns $F_{Ge}(m)=1-(1-p_0)^{m+1}$, being the probability for a first success after $\boldsymbol{m=x}$ *Bernoulli* experiments. The probability $\boldsymbol{p_0}$ for a success in each such experiment must be specified in **J**. Start reading here for more: *http://en.wikipedia.org/wiki/Geometric_distribution* . 

[84] Binom$_P$ equals BINOMDIST(***g; n; p₀; 0***) and Binom equals BINOMDIST(***m; n; p₀; 1***) in MS Excel.

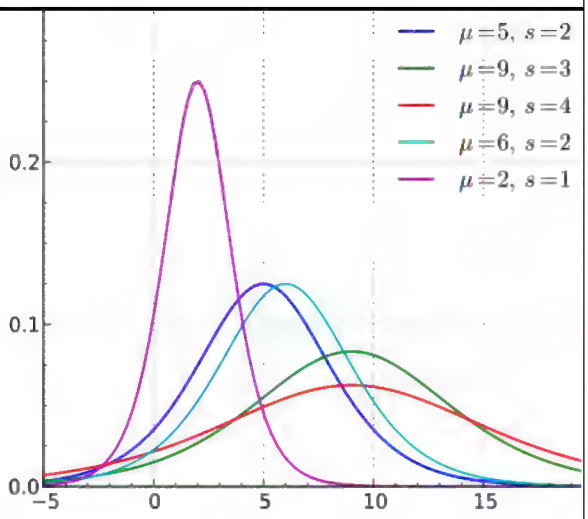
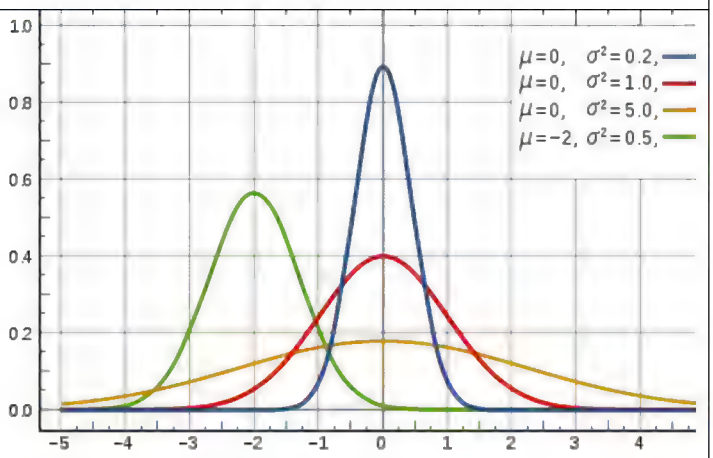
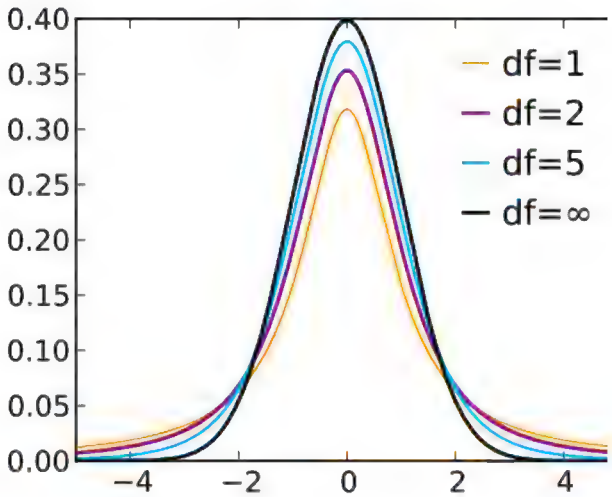
[85] The exact result for said probability is 0.626, calculated using a mathematically more elaborated model of such a test. These results show nicely that two significant digits are a typical accuracy of statistical statements – frequently the (simplified) statistical model used matches reality no better than that.

Name	Remarks (see p. 74 for general information)
Poiss, Poisλ	*Poisson distribution* with the *number of successes* ***g*** in **X** and either the *Poisson parameter* ***λ*** in **J** (for Poisλ) or the *gross probability of a success* $\boldsymbol{p_0}$ in **J** and the *sample size* ***n*** in **K**. (for Poiss). In the latter case, $\lambda = n \cdot p_0$ is calculated automatically.  Poisλ$_P$ [86] computes $P_P(g;\lambda) = \frac{\lambda^g}{g!} e^{-\lambda}$, and Poisλ returns the *cdf* for the maximum number of successes ***m*** in **X**. The *Poisson distribution* provides the mathematically easiest statistical model for industrial sampling tests. Repeating the example we made with Binom above, we get 0.638 here. Read here for more information: *http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm* .
Cauch	*Cauchy-Lorentz distribution* (also known as *Lorentz* or *Breit-Wigner distribution*) with the *location* $\boldsymbol{x_0}$ specified in **J** and the *shape* $\boldsymbol{\gamma}$ in **K** :  Cauch$_P$ returns $$f_{Ca}(x) = \left\{ \pi\gamma \cdot \left[1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right] \right\}^{-1},$$ Cauch returns $$F_{Ca}(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x - x_0}{\gamma} \right),$$ Cauch^{-1} returns $F_{Ca}^{-1}(p) = x_0 + \gamma \tan\left[\pi \cdot \left(p - \frac{1}{2} \right) \right]$. This distribution is quite popular in physics. It is a special case of *Student's t-distribution*. Start reading here for more: *http://en.wikipedia.org/wiki/Cauchy_distribution* .

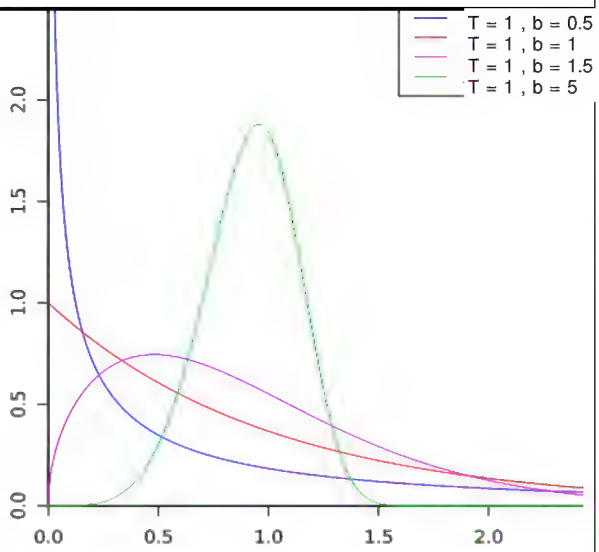
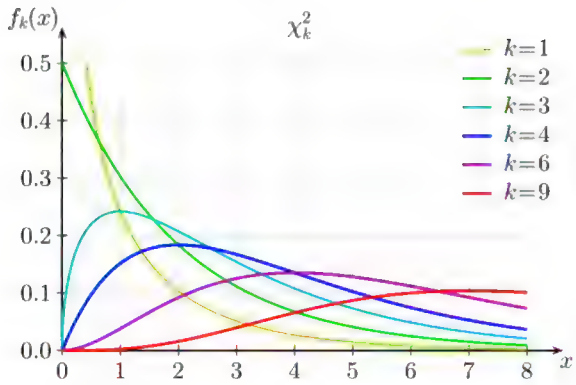
[86] Poisλ$_P$ corresponds to POISSON(***g; λ; 0***) and Poisλ to POISSON(***g; λ; 1***) in MS Excel.

Name	Remarks (see p. 74 for general information)
Expon	*Exponential distribution* with the *rate* λ in **J**. Expon$_P$[87] returns $f_{Ex}(x) = \lambda \cdot e^{-\lambda x}$. See some curves plotted here. Expon returns $F_{Ex}(x) = 1 - e^{-\lambda x}$. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3667.htm . 
F(x)	*Fisher's F-distribution* with the *degrees of freedom* in **J** and **K**. It is used e.g. for the analysis of variance (ANOVA). The picture shows the *pdf* plotted for different *degrees of freedom* ***m*** *and* ***n*** corresponding to ***j*** and ***k***. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3665.htm . 
LgNrm	*Log-normal distribution* with $\mu = \ln \bar{x}_g$ specified in **J** and $\sigma = \ln \varepsilon$ in **K**. LgNrm$_P$ returns $$f_{Ln}(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{[\ln(x)-\mu]^2}{2\sigma^2}} .$$ Some curves are plotted here. LgNrm returns $F_{Ln}(x) = \Phi\left(\frac{\ln(x) - \mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standard normal cdf* as presented at the bottom of this table. The picture shows plots of log-normal *pdf*s with the parameters specified. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3669.htm . 

[87] The pdf corresponds to EXPONDIST(*x; λ;* **0**) and the cdf to EXPONDIST(*x; λ;* **1**) in MS Excel.

Name	Remarks (see p. 74 for general information)
Logis	*Logistic distribution* with an arbitrary *mean* ***μ*** given in **J** and a *scale parameter* ***s*** in **K**. Substituting $\xi = \frac{x-\mu}{s}$, Logis$_P$ returns $f_{Lg}(x) = \frac{e^{-\xi}}{s \cdot \left(1+e^{-\xi}\right)^2}$ and Logis returns $F_{Lg}(x) = \frac{1}{1+e^{-\xi}}$. Logis^{-1} returns $F_{Lg}^{-1}(p) = \mu + s \cdot \ln\left(\frac{p}{1-p}\right)$. Start reading here for more: http://en.wikipedia.org/wiki/Logistic_distribution . 
Norml	*Normal* (or *Gaussian*) *distribution* with an arbitrary *mean* ***μ*** given in **J** and an arbitrary *standard deviation* ***σ*** in **K**. Norml$_P$[88] returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ and Norml returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the standard normal *cdf* as presented at the bottom of this table. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3661.htm . 
t(x)	Standardized *Student's t-distribution* with its *degrees of freedom* in **J** as used for hypothesis testing and calculating confidence intervals e.g. for means. The picture shows the *pdf* plotted for different *degrees of freedom*. For ***df*** → ∞ , the shoulders of the function shrink and the *pdf* approaches the one of the *standard normal distribution* (compare the red curve at NORML). Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3664.htm. 

[88] Norml$_P$ corresponds to NORMDIST(*x; μ; σ;* **0**) in MS Excel, Norml to NORMDIST(*x; μ; σ;* **1**) and Norml^{-1} to NORMINV(***F$_N$; μ; σ***) .

Name	Remarks (see p. 74 for general information)
Weibl	*Weibull distribution* with its *shape parameter* ***b*** in **J** and its *characteristic lifetime* ***T*** in **K**. Weibl$_P$[89] returns $f_W(t)=\frac{b}{T}\left(\frac{t}{T}\right)^{b-1} e^{-\left(\frac{t}{T}\right)^b}$ for $t \geq 0$ (else it returns 0). This is a very flexible function – see the curves plotted here. Weibl returns $F_W(t)=1-e^{-\left(\frac{t}{T}\right)^b}$. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm . Also see http://en.wikipedia.org/wiki/Weibull_distribution#Uses for some more application fields. 
φ(x), Φ(x)	$\varphi(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ is the *standard normal pdf* (the famous bell curve, see the red curve at Norml), while $\Phi(x)=\int_{-\infty}^{x}\varphi(\tau)d\tau$ is the corresponding *cdf* (compare the *error function* as shown on p. 208). See the link at NORML above for more information.
χ^2	The *chisquare distribution* is used for calculating confidence intervals for standard deviations and the like. The picture shows the *pdf* plotted for different *degrees of freedom*. Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3666.htm . 

[89] The *pdf* equals WEIBULL(*x; **b; T;** **0***) and the *cdf* WEIBULL(*x; **b; T;** **1***) in MS Excel.

More Statistical Formulas

Note that complete results of measured samples must include both an information about the expected value and about its uncertainty.

- For samples drawn out of a Gaussian (additive) process, the expected value is the arithmetic mean and its uncertainty is given by its standard error (see $\bar{x}$ and SERR).
- For samples drawn out of a log-normal (multiplicative) process, the expected value is the geometric mean and its uncertainty is given by its scattering factor (see $\bar{x}_g$, ε_m).
- For samples drawn out of other processes other measures apply.

Be assured not everything is Gaussian in real world! [90] Process features can be detected (and should be checked well in advance of calculating e.g. means) using suitable tests – turn to applicable statistical reference literature.

The following functions as named in the left column are all found in STAT (except $\bar{x}$ and s, which are added to this table for comparison only).

Name	Remarks (see p. 74 for general information)
CORR	(1) For a linear fit model, the *correlation coefficient* is $r = \frac{s_{xy}}{s_x \cdot s_y}$. See s_{XY} and s below. For an arbitrary fit model $R(\boldsymbol{x})$, the value $r^2 = 1 - \frac{\sum[R(x_i) - y_i]^2}{\sum(\bar{y} - y_i)^2}$ is the *coefficient of determination*; it indicates the fraction of the variation of the dependent data $\boldsymbol{y}$ determined by the variation of the independent data $\boldsymbol{x}$. For $r^2 = 1$, $\boldsymbol{y}$ is fully determined by $\boldsymbol{x}$; for $r^2 = 0$, $\boldsymbol{y}$ is completely independent of $\boldsymbol{x}$; and for e.g. $r^2 = 0.93$, 93% of the total variation of $\boldsymbol{y}$ is due to $\boldsymbol{x}$. A regression is *significant* if $\lvert r \rvert \cdot \sqrt{\frac{n-2}{1-r^2}} > t_{n-2}^{-1}(0.99)$, with the right side being the inverse of the *t*-distribution.
COV	(1) For a linear fit model, the *population covariance* is $COV_{xy} = \left(n\sum x_i y_i - \sum x_i \sum y_i\right)/n^2$. Compare s_{XY} below.

[90] Generally, the statistical model shall be chosen that matches observations best. In many real life cases, however, dramatic deviations from the model distribution are found – then you cannot expect the calculated consequences matching the reality any better.
By the way: Since the *pdf* of the Gaussian distribution will never reach zero, this statistical model tells you to expect individual items far, far away from the mean value when your sample becomes large enough. This, however, does not match reality. So we must conclude nothing at all is really Gaussian in real world. Nevertheless, the Gaussian distribution is a very successful model for describing a lot of real world observations. Just never forget the limits of such models.

Name	Remarks (see p. 74 for general information)
L.R.	(2) For a linear fit model, the line parameters are $a_0 = \frac{\sum x_i^2 \cdot \sum y_i - \sum x_i \cdot \sum x_i y_i}{n \cdot (n-1) \cdot s_x^2}$ and $a_1 = \frac{s_{xy}}{s_x^2} = r \cdot \frac{s_y}{s_x}$ (see CORR above, s_{XY} and s below). Their *standard errors* can be calculated using $s_E(a_1) = \frac{s_y}{s_x}\sqrt{\frac{1-r^2}{n-2}}$ and $s_E(a_0) = s_E(a_1) \cdot \sqrt{\frac{n-1}{n} s_x^2 + \bar{x}^2}$. Generally, a regression parameter is *significant* if $\left\lvert a_i \right\rvert / s_v(a_i) > t_{n-2}^{-1}(0.995)$, with the right side being the inverse of the *t*-distribution.
s_{XY}	(1) For a linear fit model, the *sample covariance* is $s_{xy} = \left(n\sum x_i y_i - \sum x_i \sum y_i\right) / [n \cdot (n-1)]$. Compare COV above.
s, SERR	(2) The *sample variance* is $s_x^2 = \frac{n\sum x_i^2 - \left(\sum x_i\right)^2}{n \cdot (n-1)} = \frac{\sum x_i^2 - n \cdot \bar{x}^2}{n-1}$. The *sample standard deviation* (*SD*) is $s_x = +\sqrt{s_x^2}$. And the *standard error* (i.e. the *SD* of the *mean* $\bar{x}$) is $s_{Ex} = s_x / \sqrt{n}$.
s_W , $SERR_W$	(1) The *sample SD* for <u>weighted</u> data (where the weight y_i of each data point x_i was entered via [Σ+]) is $s_w = +\sqrt{\frac{\sum y_i \cdot \sum\left(y_i \cdot x_i^2\right) - \left[\sum\left(y_i \cdot x_i\right)\right]^2}{\left(\sum y_i - 1\right) \cdot \sum y_i}}$. And the respective *standard error* (the *SD* of the *mean* $\bar{x}_W$) is $s_{Ew} = +\frac{1}{\sum y_i} \cdot \sqrt{\frac{\sum y_i \cdot \sum\left(y_i \cdot x_i^2\right) - \left[\sum\left(y_i \cdot x_i\right)\right]^2}{\sum y_i - 1}}$.
$\bar{x}$	(2) The *arithmetic mean* (or *average*) is calculated as $\bar{x} = \frac{1}{n}\sum x_i$.
$\bar{x}_g$	(2) The *geometric mean* is calculated as $\bar{x}_g = \sqrt[n]{\prod x} = e^{\frac{1}{n}\sum \ln x}$.
$\bar{x}_w$	(1) The *arithmetic mean* for <u>weighted</u> data (see s_W) is calculated as $\bar{x}_w = \frac{\sum x_i y_i}{\sum y_i}$.
ε	(2) The *scattering factor* ε_x for a sample of log-normally distributed data is calculated via: $\ln(\varepsilon_x) = \sqrt{\frac{\sum \ln^2(x_i) - 2n \cdot \ln(\bar{x}_g)}{n-1}}$. Compare s.

Name	Remarks (see p. 74 for general information)
ε_m	(1) The *scattering factor* of the *geometric mean* is $\varepsilon_m = \varepsilon^{1/\sqrt{n}}$. Compare SERR.
ε_p	(2) The *scattering factor* ε_p for a population of log-normally distributed data is calculated via: $\ln(\varepsilon_p) = \sqrt{\frac{n-1}{n}} \cdot \ln(\varepsilon_x)$. Compare σ.
σ	(2) The *SD* of the population is $\sigma_x = +\frac{1}{n} \cdot \sqrt{\sum (x_i - \bar{x})^2} = s_x \cdot \sqrt{\frac{n-1}{n}}$.
σ_W	(1) The *SD* of the population for weighted data (see s_W) is $\sigma_w = +\sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$.

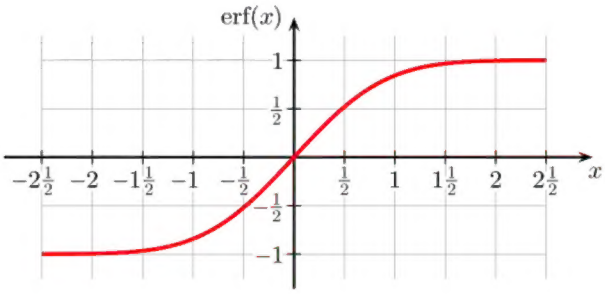
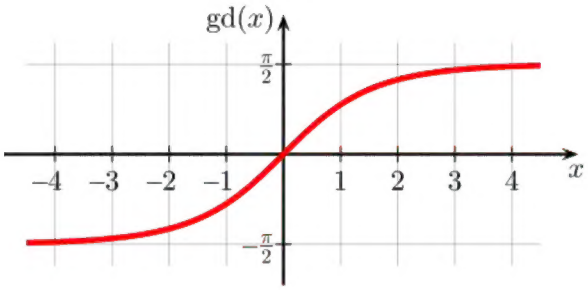
Orthogonal Polynomials

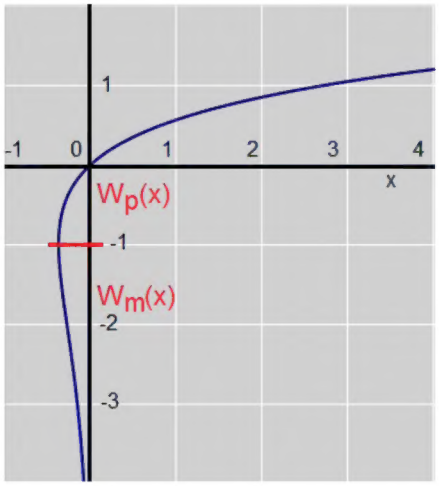
These polynomials are all found in X.FCN.

Name	Remarks (see p. 74 for general information)
H_n	(2) *Hermite polynomials* for probability: $H_n(x)=(-1)^n\cdot e^{x^2/2}\cdot\frac{d^n}{dx^n}\left(e^{-x^2/2}\right)$ with ***n*** in **Y**, solving the differential equation $f''(x)-2x\cdot f'(x)+2n\cdot f(x)=0$.
H_{np}	(2) *Hermite polynomials* for physics: $H_{np}(x)=(-1)^n\cdot e^{x^2}\cdot\frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in **Y**, solving the same differential equation.
L_n	(2) *Laguerre polynomials* (compare $L_n\alpha$ below): $L_n(x)=\frac{e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^n e^{-x}\right)=L_n^{(0)}(x)$ with ***n*** in **Y**, solving the differential equation $x\cdot f''(x)+(1-x)f'(x)+n\cdot f(x)=0$.
$L_n\alpha$	(3) *Laguerre's generalized polynomials* (compare L_n above): $L_n^{(\alpha)}(x)=\frac{x^{-\alpha}e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^{n+\alpha}e^{-x}\right)$ with ***n*** in **Y** and **α** in **Z**.
P_n	(1) *Legendre polynomials*: $P_n(x)=\frac{1}{2^n n!}\cdot\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]$ with ***n*** in **Y**, solving the differential equation $\frac{d}{dx}\left[\left(1-x^2\right)\cdot\frac{d}{dx}f(x)\right]+n(n+1)f(x)=0$.
T_n	(2) *Chebychev* (a. k. a. *Čebyšev, Tschebyschow, Tschebyscheff*) *polynomials of first kind* $T_n(x)$ with ***n*** in **Y**, solving the differential equation $\left(1-x^2\right)f''(x)-x\cdot f'(x)+n^2\cdot f(x)=0$.
U_n	(2) *Chebychev polynomials of second kind* $U_n(x)$ with ***n*** in **Y**, solving the differential equation $\left(1-x^2\right)f''(x)-3x\cdot f'(x)+n(n+2)f(x)=0$.

Even More Mathematical Functions

Also these are all found in X.FCN. Some of them are for pure mathematics only, but were useful at some stage of the *WP 34S* project, so we made them accessible for the public.

Name	Remarks (see p. 74 for general information)
AGM	(2) Returns the *arithmetic-geometric mean*. Find more about it here: http://mathworld.wolfram.com/Arithmetic-GeometricMean.html .
erf	(1) Returns the *error function* $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-\tau^2}d\tau$. Note that $erf\left(\frac{x}{\sqrt{2}}\right)=2\cdot\Phi(x)-1$ with $\Phi(x)$ representing the *standard normal cdf* as described on p. 203. 
erfc	(1) Returns the *complementary error function* $erfc(x)=1-erf(x)=\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-\tau^2}d\tau$. This function is related to the *error probability* of the *standard normal distribution*.
g_d	(1) Returns the *Gudermann function* $g_d(x)=\int_0^x\frac{d\xi}{\cosh\xi}$. It links hyperbolic and trigonometric functions. See the plot for its real values. The *inverse Gudermann function* is $g_d^{-1}(x)=\int_0^x\frac{d\xi}{\cos\xi}$. Start reading here for more: http://en.wikipedia.org/wiki/Gudermannian_function . 
Iβ	(3) Returns the *regularized (incomplete) beta function* $\frac{\beta_x(x,y,z)}{\beta(y,z)}$ with $\beta(y,z)$ being *Euler's Beta* (see there) and $\beta_x(x,y,z)=\int_0^x t^{y-1}(1-t)^{z-1}dt$ being the *incomplete beta function*.
$I\Gamma_p$	(2) Returns the *regularized (incomplete) gamma function* $P(x,y)=\frac{\gamma(x,y)}{\Gamma(x)}$. For $\gamma(x,y)$ see γ_{XY} below, for $\Gamma(x)$ see there.
$I\Gamma_q$	(2) Returns the *regularized (incomplete) gamma function* $Q(x,y)=\frac{\Gamma_u(x,y)}{\Gamma(x)}$. For $\Gamma_u(x,y)$ see Γ_{XY} below, for $\Gamma(x)$ see there.

Name	Remarks (see p. 74 for general information)
W_p, W_m	(1) Returns *Lambert's W* with its principal branch (called W_p here) and its negative branch (called W_m for minus). The connecting point is (-1/e , -1). The diagram shows the real values of both branches. Start reading here for more information: *http://en.wikipedia.org/wiki/Lambert_W_function* . Learn more here: *http://mathworld.wolfram.com/LambertW-Function.html* . 
γ_{XY}	(2) Returns the *lower incomplete gamma function* $\gamma(x,y)=\int_0^y t^{x-1}e^{-t}dt$. Required for $I\Gamma_p$ above.
Γ_{XY}	(2) Returns the *upper incomplete gamma function* $\Gamma_u(x,y)=\int_y^\infty t^{x-1}e^{-t}dt$. Required for $I\Gamma_q$ above.
ζ	(1) Returns *Riemann's Zeta* for real arguments, with $\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$ for $x>1$, and its analytical continuation for $x<1$: $\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x)$. Read here for more: *http://mathworld.wolfram.com/RiemannZetaFunction.html* .

You will find lots of information about the special functions implemented in your *WP 34S* in the internet in addition. Generally, *Wikipedia* is a good starter – check the articles in different languages since they may well contain different material and use different approaches. *Mathworld* may contain more details than you ever wanted to know. And for applied statistics, the *NIST Sematech* online handbook quoted above is a competent source. Further references are found at these sites.

APPENDIX J: RELEASE NOTES

	Date	Release notes
1	9.12.08	Start
1.1	15.12.08	Added the table of indicators; added NAND, NOR, XNOR, RCLWS, STOWS, //, N, SERR, SIGMA, < and >; deleted HR, INPUT, 2 flag commands, and 2 conversions; extended explanations for addressing and COMPLEX & …; put XOR on the keyboard; corrected errors.
1.2	4.1.09	Added ASRN, CBC?, CBS?, CCB, SCB, FLOAT, MIRROR, SLN, SRN, >BIN, >DEC, >HEX, >OCT, BETA, D>R, DATE, DDAYS, D.MY, M.DY, Y.MD, CEIL, FLOOR, DSZ, ISZ, D>R, R>D, EMGAM, GSB, LNBETA, LNGAMMA, MAX, MIN, NOP, REAL, RJ, W and WINV, ZETA, %+ and %-; renamed the top left keys B, C, and D, and bottom left EXIT.
1.3	17.1.09	Added AIP, ALENG, ARCL, AROT, ASHF, ASTO, ATOX, XTOA, AVIEW, CLA, PROMPT (all taken from 42S), CAPP, FC?C, FS?C, SGMNT, and the …# commands; renamed NBITS to BITS and STOWS to WSIZE; specified the bit commands closer; deleted the 4 carry bit operations.
1.4	10.2.09	Added CONST and a table of constants provided, D>J and J>D, LEAP?, %T, RCL and STO ▲ and ▼, and 2 forgotten statistics registers; deleted CHS, EMGAM, GSB, REAL and ZETA; purged and renamed the bit operations; renamed many commands.
1.5	5.3.09	Added RNDINT, CONV and its table, a memory table, the description of XEQ B, C, D to the operation index, and *a* and g_e to the table of constants; put CLSTK on a key, moved CLΣ and FILL, changed the % and log labels on the keyboard, put CLALL in X.FCN; checked and cleaned alpha mode keyboard and added a temporary alpha keyboard; rearranged the alphabet to put Greek after Latin, symbols after Greek consistently; separated the input and non-programmable commands; cleaned the addressing tables.
1.6	12.8.09	Added BASE, DAYS+, DROP, DROPY, E3OFF, E3ON, FC?F, FC?S, FIB, FS?F, FS?S, GCD, LCM, SETDAT, SETTIM, SET24, SINC, TIME, VERS, αDAY, αMONTH, αRC#; %Σ, as well as F-, t-, and χ^2-distributions and their inverses; reassigned DATE, modified DENMAX, FLOAT, αROT, and αSHIFT; deleted BASE arithmetic, BIN, DEC, HEX, and OCT; updated the alpha keyboards; added flags in the memory table; included indirect addressing for comparisons; added a paragraph about the display; updated the table of indicators; corrected errors.
1.7	9.9.09	Added P.FCN and STAT catalogs, 4 more conversions, 3 more flags, Greek character access, CLFLAG, DECOMP, DENANY, DENFAC, DENFIX, Iβ, IΓ, αDATE, αRL, αRR, αSL, αSR, αTIME, 12h, 24h, fraction mode limits, normal distribution and its inverse for arbitrary μ and σ, and Boolean operations working within FLOAT; deleted αROT, αSHIFT, the timer, and forced radians after inverse hyperbolics; renamed WINV to W^{-1}, and beta and gamma commands to Greek; added tables of catalog contents; modified label addressing; relabeled PRGM to P/R and PAUSE to PSE; swapped SHOW and PSE as well as Δ% and % on the keyboard; relabeled Q; corrected CEIL and FLOOR; updated X.FCN and alpha commands; updated the virtual alpha keyboard.
1.8	29.10.09	Added R-CLR, R-COPY, R-SORT, R-SWAP, RCLM, STOM, alpha catalogs, 1 more constant and some more conversions, a table of error messages, as well as the binomial, Poisson, geometric, Weibull and exponential distributions and their inverses; renamed some commands; put √‾ instead of π on hotkey D.
1.9	14.12.09	Added two complex comparisons; swapped and changed labels in the top three rows of keys, dropped CLST; completed function descriptions in the index.
1.10	19.1.10	Added IMPFRC, PROFRC, ^{C}ENTER, αBEG, αEND, and an addressing table for items in catalogs; updated temporary alpha mode, display and indicators, RCLM and STOM, alpha-commands and the message table; renamed the exponential distribution; wrote the introduction.
1.11	21.9.10	Changed keyboard layout to bring Π and Σ to the front, relabeled binary log, swapped the locations of π, CLPR, and STATUS, as well as SF and FS?; created a menu TEST for the comparisons removed and the other programmable tests from P.FCN; added %MG, %+MG, %MRR, RESET, SSIZE4, SSIZE8, SSIZE?, ^{C}DROP, ^{C}FILL, ^{C}R↓, ^{C}R↑, registers J and K, a table of contents and tables for stack mechanics and addressing in complex operations; updated memory and real number addressing tables, DECOMP, αOFF, αON, Π, and Σ; renamed ROUNDI, WSIZE?, β(x,y), Γ(x) and the constant p_0 ; deleted DROPY (use x↔y, DROP instead), αAPP, αBEG, αEND, and the "too long error" message; deleted Josephson and von Klitzing constants (they are just the inverses of other constants already included in CONST); brought more symbols on the alpha keyboard.
1.12	22.12.10	Modified keyboard layout; added catalogs MODE and PROB; changed mode word, catalog contents and handling (XEQ instead of ENTER), as well as some non-programmable info commands; expanded IMPFRC and PROFRC; added a paragraph about the fonts provided and explained alpha catalogs in detail; added PRIME? and some conversions; deleted FRACT, OFF and ON.
1.13	3.2.11	Modified keyboard layout; modified αTIME, radix setting, H.MS+ and H.MS-; added EVEN?, FP?, INT?, LZOFF, LZON, ODD?, RCLS, STOS, returned FRACT; added and renamed some conversions; updated the paragraph about display; added appendices A and B; baptized the device *WP 34S*.

1.14	18.3.11	Started the Windows emulator. Added DEC and INC, renamed FLOAT to DECM; redefined αTIME and H.MS mode; updated appendix A; documented the annunciators BEG and = as well as underflows and overflows in H.MS; corrected some errors showing up with the emulator.
1.15	21.3.11	Modified FIX, removed ALL from MODE, updated CONV.
1.16	27.3.11	Added LBL?, f'(x), and f''(x); modified PSE; upgraded catalog searching.
1.17	9.5.11	Modified keyboard layout for adding a fourth hotkey; added AGM, BATT, B_n, B_n*, Cauch, Lgnrm, Logis and their inverses, all the pdf, COV, CUBE, CUBERT, DEG→, ENGOVR, ENTRY?, erfc, GRAD→, GTO . *hotkey*, KEY?, RAD→, SCIOVR, SERRw, SLVQ, sw, sxy, TICKS, TVM, xg, ε, ε_m, ε_p, ζ, σw, $(-1)^X$, the polynomials, four angular conversions, four Planck constants, the regional settings, global alpha labels, and three messages; renamed most cdf; changed →DEG, →RAD, →GRAD to leaving angular mode as set; altered PSE for early termination by key-stroke; made D.MY default instead of Y.MD; moved degrees to radians conversions to CONV; removed ^{C}CLx, H.MS mode, %+ and %-; corrected errors.
1.18	5.6.11	Expanded program memory; modified label addressing (A ≠ 'A') and fraction mode limits, changed ANGLE to work in real and complex domains, renamed MOD to RMDR, changed the keyboard layout; put BACK, ERR, SKIP, and SPEC? to the main index; added CAT and the I/O commands for flash memory, expanded R-COPY; corrected x→α.
2.0	21.7.11	Entered beta test phase. Added DAY, MONTH, YEAR, FAST, SLOW, S.L, S.R, VWα+, flag A, ON + and –, some constants, and a paragraph about I/O; renamed old DAY to WDAY, RRCL to RCFRG, SRCL to RCFST; added an inverse conversion shortcut, stones⇆kg, and changed Pa⇆mbar to Pa⇆bar; modified the VIEW commands, ALL, DISP, MODE, RCLM, STOM, and X.FCN; repaired hyperlinks; corrected some errors; included flash.txt; updated the first chapters, explained stack mechanics in more detail.
2.1	3.10.11	Added serial I/O commands, DEL_P, DSL, EXPT, IBASE?, INTM?, ISE, KTY?, MANT, NEXTP, PUTK, REALM?, RM, RM?, SMODE?, TOP?, $^x\sqrt{y}$, signed tests for zero, some constants, and the paragraph about interactive programming; updated the values in CONST to CODATA 2010, also updated SLVQ, SHOW, Σ, Π, and the paragraphs about statistics, predefined alpha labels and memory; corrected some errors; deleted complex ANGLE, →BIN, →DEC, →HEX, and →OCT; redistributed the contents of X.FCN and P.FCN; renamed S.L and S.R to SDL and SDR; put '?' on the alpha keyboard and moved £ to P to make room for π; expanded Appendix A; reorganized the structure of the document; added first aid to the front page; rewrote the keyboard chapter.
2.2	1.11.11	Added MSG, y↔, z↔, and matrix operations, a paragraph about them and two new error messages for them, plus a footnote for DEL_P; updated the introduction to statistics. This version is the last one working with the old overlays. It is maintained to incorporate the latest common bug fixes – last v2.2 build available is 2739 so far.
3.0	21.4.12	Added CLPALL, CROSS, DOT, iRCL, sRCL, END, FLASH?, g_d, g_d^{-1}, GTO.▲ and ▼, LOAD…, LocR, LocR?, MEM?, NEIGHB, PopLR, RDP, REGS, REGS?, RSD, SEPOFF, SEPON, SETJPN, STOPW, t↔, ULP, ↔, #, as well as SUMS and MATRIX catalogs and four conversions; renamed CLFLAG to CFALL, CUBE to x^3 and CUBERT to $^3\sqrt{x}$, KTY? to KTP?, and αVIEW to VIEWα; split Lambert's W into W_p and W_m; returned →BIN, →HEX, and →OCT; made PSTO and SAVE nonprogrammable; redefined SHOW, corrected CLREG, deleted DEL_P; changed keyboard layout to bring MATRIX, CLP, SF, and CF to the front and to swap OFF and SHOW, removed x↔α from the key plate and π from CONST; modified the virtual alpha keyboard, some characters and the respective catalogs; redistributed commands in the catalogs; updated and rearranged large parts of the text; added information about complex calculations, bitwise integer operations, browsers, local data, memory management, and the character sets; implemented five new ttf-fonts.
3.1	30.11.12	Added the print commands 🖶…, graphic commands g…, error probabilities, ½, IDIV, $I\Gamma_q$, MOD, XTAL?, γ_{XY}, Γ_{XY}, ×MOD, ^MOD, four conversions, two messages, an optional back label, and some references to external documents; modified GCD and LCM, DATE, SETDAT, SETTIM, and TIME; expanded STOPW; renamed IBASE? to BASE? and IΓ to $I\Gamma_p$; purged TEST moving non-tests into P.FCN; abandoned the internal expert's catalog; updated and rearranged large parts of the introductory text and the appendices, also explaining the floating point formats, carry and overflow conditions in integer modes, the emulator, troubleshooting, corresponding operations of *HP-42S and HP-16C*, and some hardware modifications; moved most mathematical explanations into Appendix I; modified many internal page number references to ease working with a printed manual.